Android Apps in

Do-It Yourself Approach

to

Learning Mobile App Development in Android

May 2025

**Overview**

Learning is a continuous life-long process in the software development and design industry and change happens so fast it makes it very difficult to keep pace with it.

Android Nougat, Marshmallow, Lollipop bring us great opportunities, several issues and a requirement to learn rapidly and implement just as fast.

Android Wear is the next big thing happening and will change very rapidly for the simple arrival of the Bluetooth new version 5 and other similar developments.

Android Auto and Wear bring two big income generating opportunities to the new age entrepreneur and full time developer to contribute to the worldwide market and make an impact that could benefit the innovator right now.

Imagine 80% of the mobile devices shipped and sold are Android devices today which opens a big money spinning environment for the people with initiative to explore and exploit it.

Simple, elegant apps with UI pleasing to the audience that actually deliver results is the hot demand right now.

So whatever your background with Android App development made so easy to learn and practice the world is at your feet for taking.

**Table of Contents**

1. Cover Page
2. Overview
3. Chapter One Overview of Android
4. Chapter Two Main Building Blocks of Android
5. Chapter Three Graphical User Interface, Views, Layouts , Dynamic UI
6. Chapter Four Part I Android Services, Android IPC & RPC, AIDL, APIs
7. Chapter Four Part II AsyncTask, JobScheduler, Loader, Handler
8. Chapter Five Android Threading, Multiple Threads, Thread Lifecycle
9. Chapter Six Hardware Interface and Sensors
10. Chapter Seven Networking, Web Service, Remote APIs
11. Chapter Eight - Part One Maps, Location and Activity APIs

Chapter Eight - Part Two String Localization and Internationalization

1. Chapter Nine Data Storage Methods
2. Chapter Ten In-App Billing
3. Chapter Eleven Push Messages
4. Chapter Twelve Communicating with Remote Devices
5. Chapter Thirteen How to Write a Test Case and Use it in Android

TDD (Test Driven Development in Android)

1. Chapter Fourteen Google Play Services and Distributing the App
2. Chapter Fifteen Using Dependencies as Picasso, Butterknife, Cupboard
3. Additional Information Tips and Tricks for the Developer
4. Conclusion

**ample Projects Illustrated in Android Studio for Beginners**

**Design Stage**

* https://developer.android.com/training/**design-navigation**/wireframing.html
* https://balsamiq.com/for/web-designers/?gclid=Cj0KEQjwxbDIBRCL99Wls-nLicoBEiQAWroh6qil9ysqLUDDZoRqgmjdETB5NGLePElVdaXKBTnXIWsaAi768P8HAQ
* http://wireframesketcher.com/create/android-mockups.html

**Android Manifest file**

* https://code.tutsplus.com/tutorials/android-sdk-**project-manifest**--mobile-20606
* http://androidexample.com/ - Facebook Integration

**Android Activities**

* https://medium.com/@JakobUlbrich/**building-a-settings-**screen-for-android-part-1-5959aa49337c

**Android Intents**

* http://programmerguru.com/android-tutorial/android-**intent**-example/
* https://www.mkyong.com/android/android-**activity-from-one-screen**-to-another-screen/

**Android Service**

* http://www.techotopia.com/index.php/Implementing\_an\_Android**\_Started\_Service**\_%E2%80%93\_A\_Worked\_Example
* http://inchoo.net/dev-talk/android-development/android-simple-service/

**Broadcast Receiver**

* http://www.journaldev.com/10356/android-**broadcastreceive**r-example-tutorial

**Content Provider - SharedPrefs, SQLite and Media Store**

* http://www.journaldev.com/9412/android-shared-preferences-example-tutorial
* https://examples.javacodegeeks.com/android/core/content/android-**sharedpreferences**-example/
* http://www.androidhive.info/2011/11/android-**sqlite**-database-tutorial/
* https://www.javatpoint.com/android-sqlite-tutorial
* http://mobilesiri.com/android-sqlite-database-tutorial-using-android-studio/
* https://www.codeproject.com/Articles/1031191/Android-SQLite-for-Beginners
* https://www.sandersdenardi.com/**querying-and-removing-media**-from-android-mediastore/

**Android Device Hardware - Camera, USB, Bluetooth, USB, etc.**

* https://www.tutorialspoint.com/android/android\_camera.htm
* https://code.tutsplus.com/tutorials/create-a-bluetooth-scanner-with-androids-bluetooth-api--cms-24084
* http://www.theappguruz.com/blog/android-bluetooth-connection-demo
* https://examples.javacodegeeks.com/android/core/bluetooth/bluetoothadapter/android-bluetooth-example/
* https://karanbalkar.com/2014/02/tutorial-73-working-with-bluetooth-in-android/
* http://stackoverflow.com/questions/11638216/how-to-make-an-basic-android-usb-host-application
* http://stackoverflow.com/questions/10183794/how-to-communicate-with-a-usb-device

**Android Telephony, TelephonyManager**

* https://www.javatpoint.com/android-**telephony-**manager-tutorial
* http://stackoverflow.com/questions/7514163/using-**telephony-manager**-in-android-to-find-imei-number

**Maps, Geofencing, GPRS**

* https://code.tutsplus.com/tutorials/how-to-work-with**-geofences**-on-android--cms-26639
* https://developers.google.com/android/reference/com/google/android/gms/maps/MapView

**Threading in Android**

* https://news.realm.io/news/android-**threading**-background-tasks/?gclid=COiS\_bDC2tMCFdaHaAodRIcAIQ

**Networking**

* https://developer.android.com/training/basics/**network-ops**/index.html
* https://trivedihardik.wordpress.com/2012/08/31/android-**wifi-direct**-example/

**Android Remote APIs**

* https://code.tutsplus.com/tutorials/android-sdk-making-remote-api-calls--mobile-17568
* https://www.chrisblunt.com/android-consuming-a-remote-json-api-with-volley/

**Android Web Service**

* https://www.codeproject.com/Articles/112381/Step-by-Step-Method-to-Access-Webservice-from-Andr

**Android Gestures**

* https://www.tutorialspoint.com/android/android\_**gestures**.htm

**Localization**

* https://www.icanlocalize.com/site/tutorials/android-application-**localization**-tutorial/
* http://www.ibabbleon.com/android\_app\_**localization**\_translation.html

**In-App Billing**

* https://blahti.wordpress.com/2014/07/30/how-to-**add-in-app-billing**-in-android-part-1/

**Push Messages**

* http://api.shephertz.com/tutorial/**Push-Notification**-Android/
* https://www.intertech.com/Blog/**push-notifications**-tutorial-for-android-using-google-cloud-messaging-gcm/

**Dependencies**

* http://www.journaldev.com/10439/android-**butter-knife-**example-tutorial

See additional document titled - Capgemini Android Testing and Android App Development and Cpagemini App Dev training

**Chapter One**  **Overview of Android**

Objective Understanding the concepts, terms, IDE, methods and about Android as an OS

Background

## 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 ?



Diagram 1

The Android platform is **constantly evolving and app developers** must keep up with recent trends to be able to build apps that live up to user expectations.

### Material Design

[Material design](https://www.google.com/design/spec/material-design/introduction.html) is a collection of well-thought-out design guidelines you can follow to build apps whose look and feel matches the native look and feel of Android Lollipop and Marshmallow. Owing to its simplicity and modern looks, material design enjoys a lot of popularity among both developers and designers.

### Android Wear

[Android Wear](https://www.android.com/intl/en_in/wear/) smartwatches are becoming increasingly popular because the latest models are better looking and have features, such as new and improved gestures and watch faces that make them more fun to use. Consequently, more and more developers are using the [Wear API](http://developer.android.com/wear/index.html) in their apps to make sure that their apps can interact with wearable devices.

Android provides a rich application framework that allows you to build innovative apps and games for mobile devices in a Java language environment. The documents listed in the left navigation provide details about how to build apps using Android's various APIs.

**The Android fragmentation issue and its solutions**

A term being used in the **Android** development community is **Android fragmentation**.

**Fragmentation** within **Android** is when a variety of versions of the**Android** platform, combined with a mixture of hardware result in the inability for some devices to properly run certain applications.

Back in 2007, Google and its partners in the Open Handset Alliance set out with the goal of developing open standards for smartphones. Android was to be a great equalizer, creating a level playing field for manufacturers and developers. It was a vision with clear appeal — and that’s partly why, nine years later, Android is the dominant mobile platform with a 58.75 percent share of the worldwide mobile and tablet operating systems market, according to NetMarketShare.

The open ideals that it was founded upon led to unprecedented diversity and a huge smartphone revolution, but they also sparked one of the most persistent criticisms of Android: fragmentation. There’s a fairly universal perception that Android fragmentation is a barrier to a consistent user experience, a security risk, and a challenge for app developers.  
  
We could be talking about the different versions of Android, the multitude of different devices, the manufacturer skins that sit on top, or even forks, like Amazon’s Fire OS.  
  
For ordinary folks who own a phone, fragmentation means that many Android users don’t have access to the latest and greatest features in the platform, but it has also been consistently raised as a major threat to security. Tim Cook famously referenced an article on ZDNet, entitled “Android fragmentation turning devices into a toxic hellstew of vulnerabilities” on a slide at WWDC in 2014. He even added animated flames to the word “hellstew” for a greater incendiary impact, and it played into a common perception of Android as potentially insecure.

Diagram 1 - Android Fragmentation Issue Illustrated

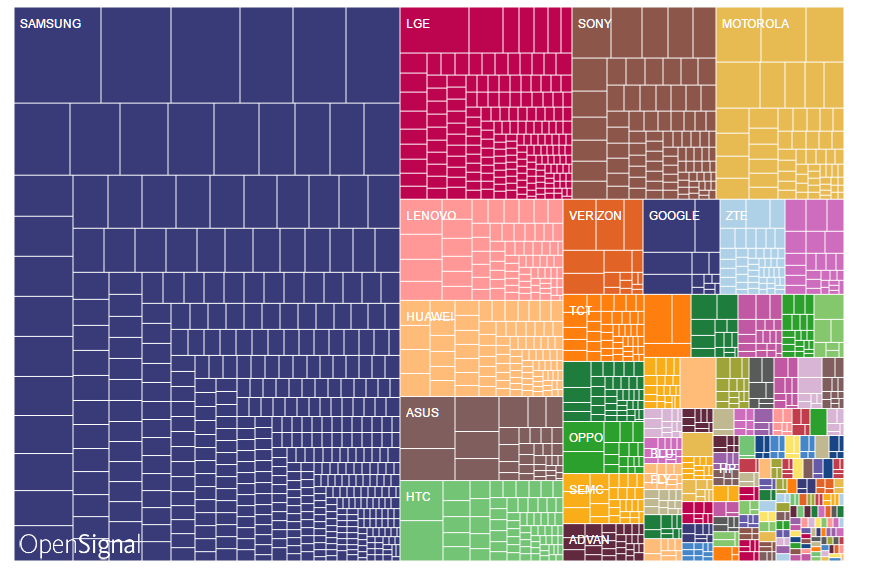


Diagram 2



References

* https://www.xda-developers.com/the-sorry-state-of-android-fragmentation/
* https://www.digitaltrends.com/mobile/what-is-android-fragmentation-and-can-google-ever-fix-it/
* http://www.techtimes.com/articles/185234/20161109/google-has-a-plan-to-solve-the-fragmentation-problem-of-android-is-android-extensions-the-answer.htm
* http://www.webopedia.com/TERM/A/Android\_fragmentation.html
* https://www.xda-developers.com/the-sorry-state-of-android-fragmentation/

**Android Studio**

Diagram 2

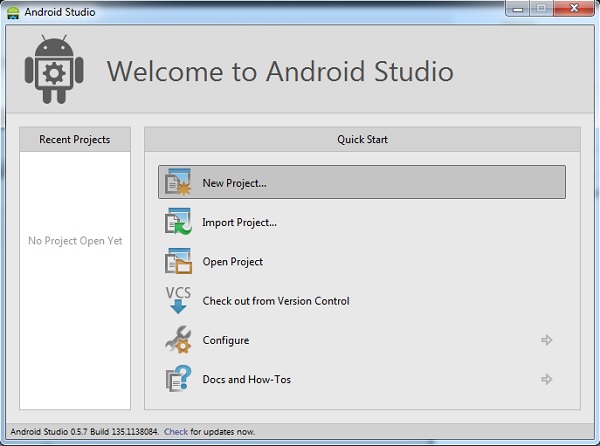


Diagram 3

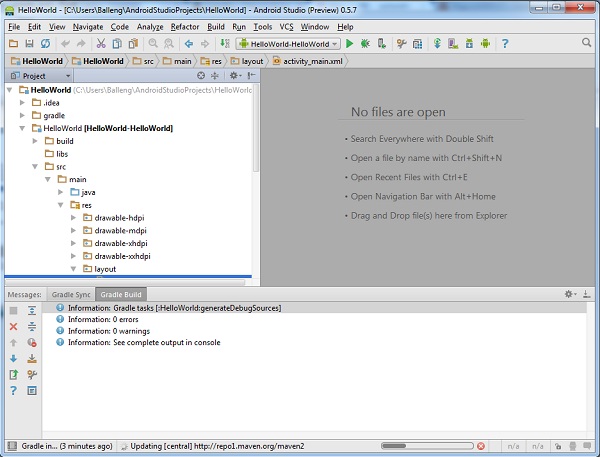


Diagram 4

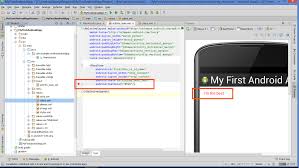


Diagram 5

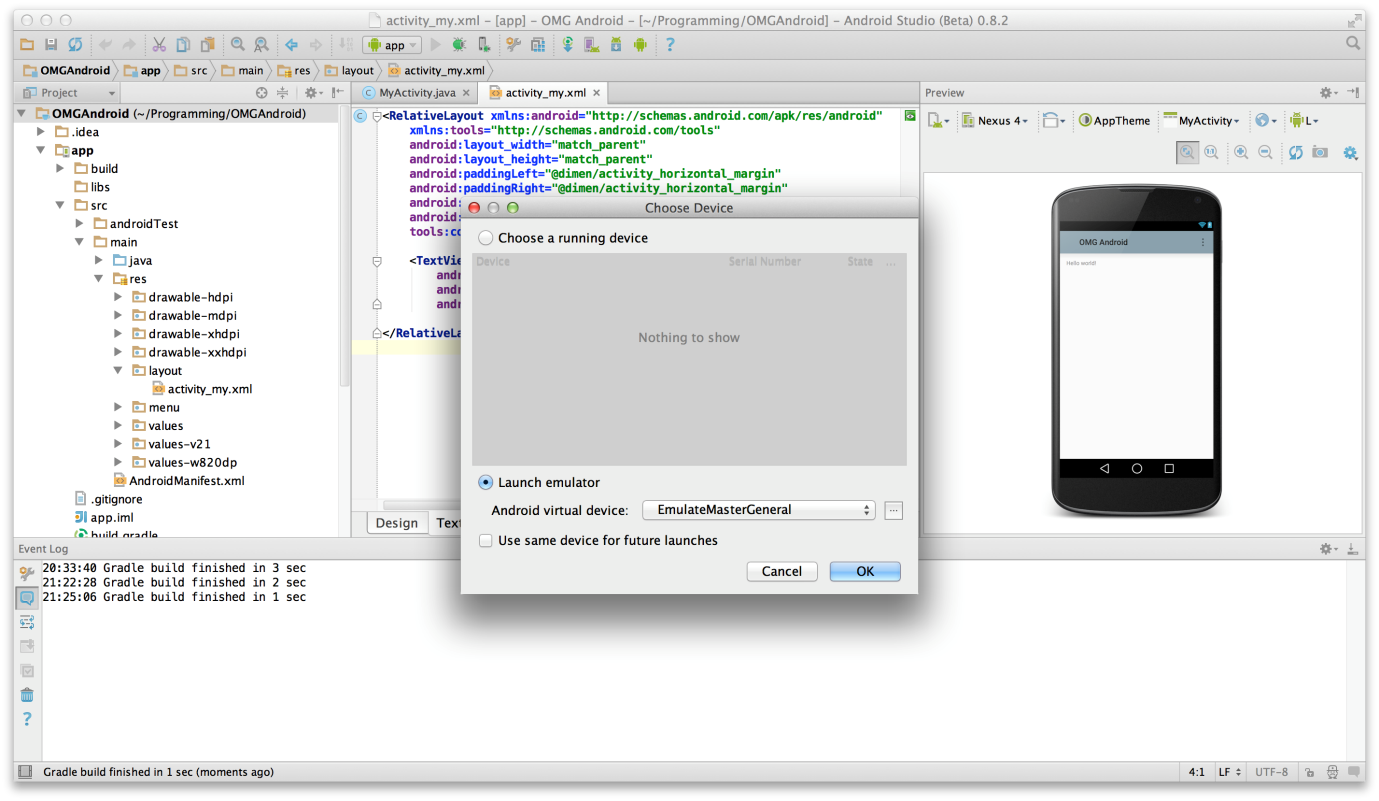


Diagram 6

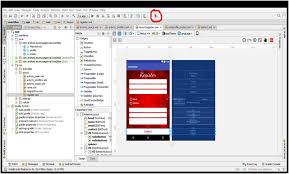
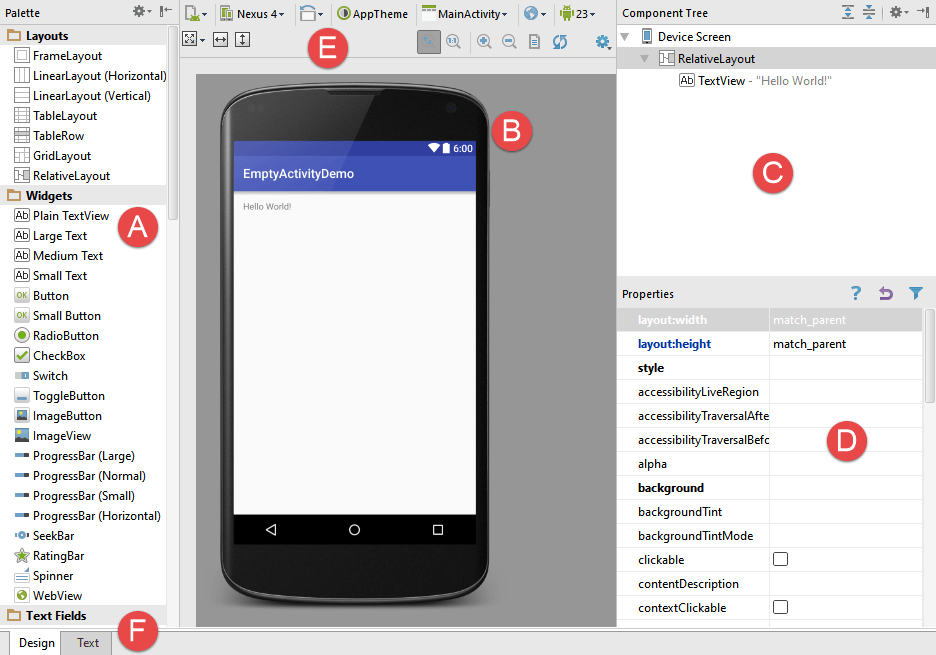


Diagram 7



**Anatomy of an Android App**

Diagram 7

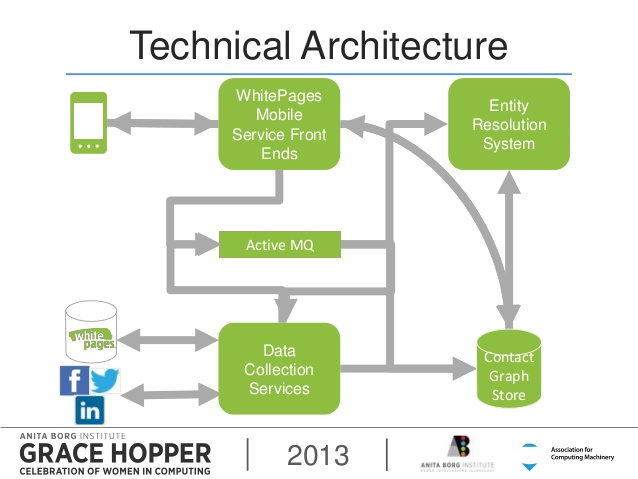
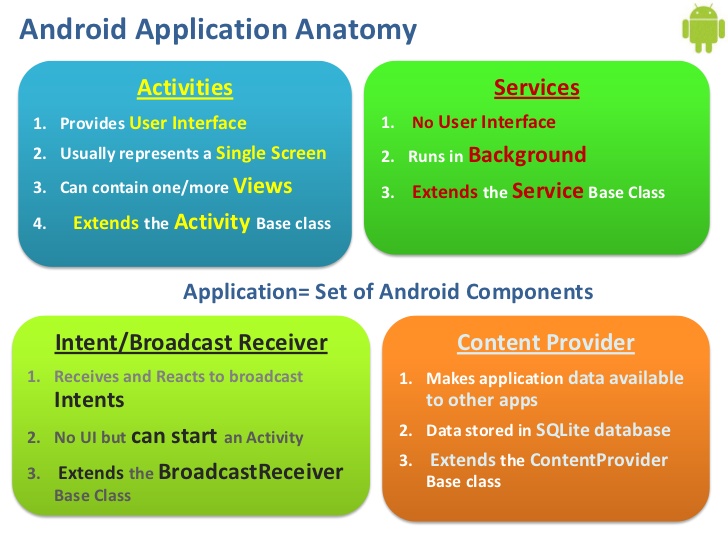


Diagram 8



**Lifecycle of an Android App**

Diagram 10

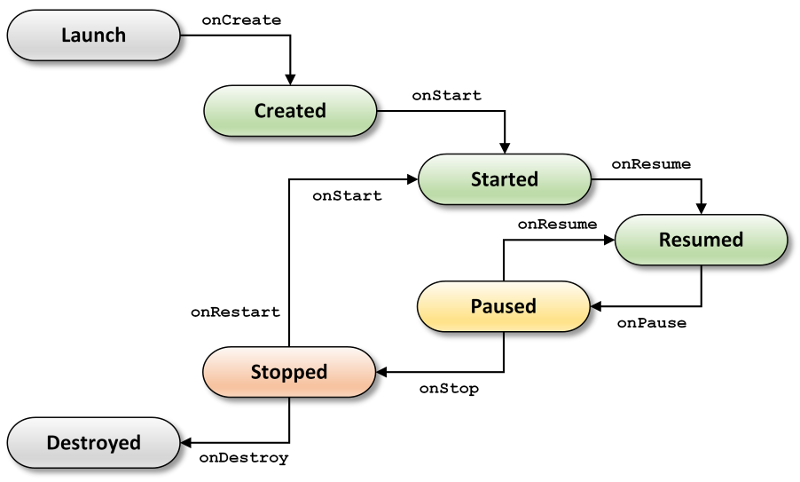


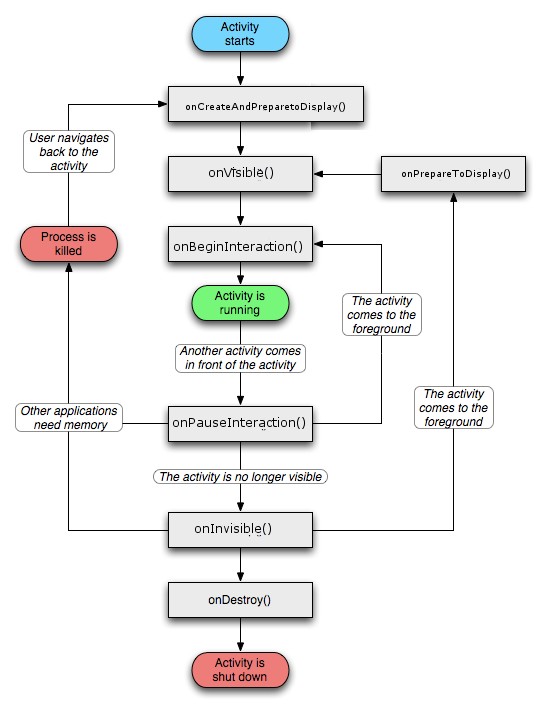
Diagram 11 **Activity Life Cycle**

Diagram 11 Fragment Life Cycle



**Setting Up the Environment**

Diagram 12

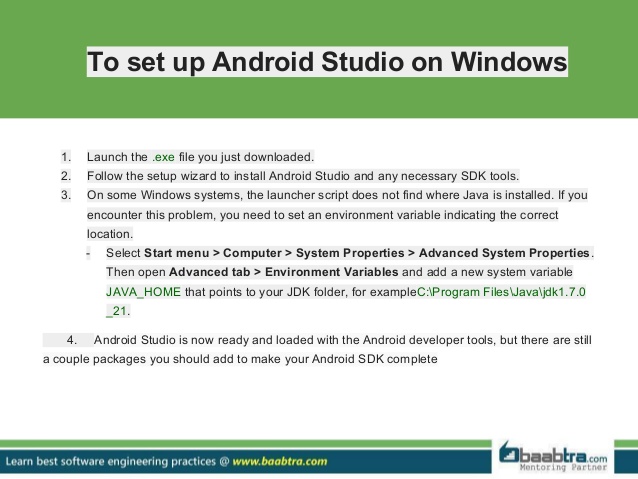
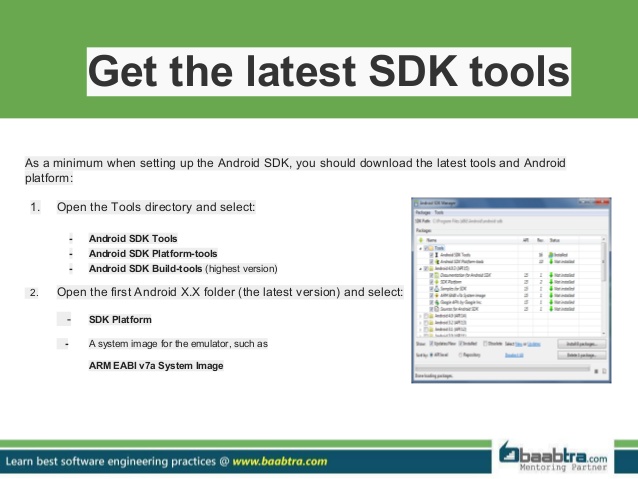


Image 1 as above

­­­Diagram 13 - Image 2 as seen below



**Setting Up the Environment**

Diagram 14 - Image 3 as seen below

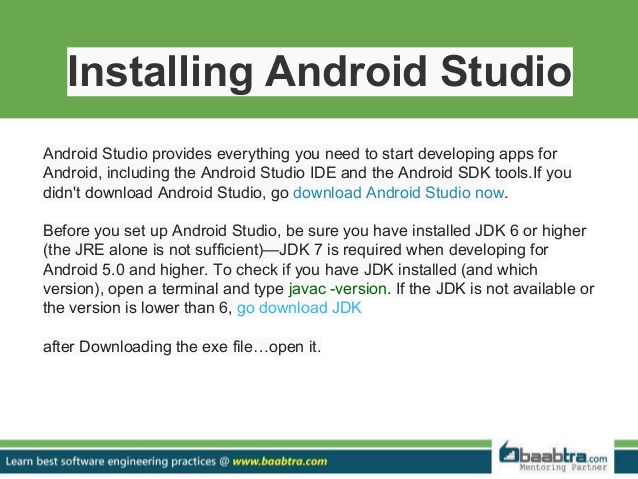


Diagram 15 as seen below



**Required software tools in Setting Up the Environment** includes:

* JDK - Java Development Kit
* Android Studio - the IDE for Android Developer
* Libraries and Dependencies from Jake Wharton like Butterknife, Picasso, Cupboard, Dagger
* Third party open source tools like
  + Design tools like Fluid UI, Droid Draw, etc.
  + Development tools like
  + Testing tools like Monkey, monkeyrunner and Mockito

Each one of us (as a team of two or as a single participant) is required to download and install JDK latest version from relevant site. As well as download and install a copy of the Android Studio version required from developer site of Google Android.

There a few plug-ins we will bring in as and when in future appropriate time.

If we are to work with Windows then JDK and Android Studio to be downloaded is for Windows as per version of Windows 7, 8, 10 or if we are to work in Linux then the appropriate Linux version of JDK and Android Studio is to be downloaded and similarly in Mac OS if we are work in El Capitan or Sierra OS then download relevant Mac versions of JDK and Android Studio please.

If participants are working from their own laptops I suggest that each one downloads and installs appropriate JDK, Android Studio and Android SDK at home in leisure before the next class.

Each learner will be required to have an Android device handy and a USB cable to connect the device to the Android Studio work system for most of our development will be tested on device and each learner can take home the resultant app or result for better learning.

With a few simple trails we will check how well each one's set up and feasible the IDE Android Studio for developer is done.

One item to check is how well you Android Studio renders its images or XML files in layout resource files. Another to check is how well the device responds to device based testing.

We need to check our debugging tools like DDMS, ADB, Logcat and all work well.

The next test is very important we need to test how well our Android Studio IDE work area responds with intellisense information as we work.

Remember this is participative and interactive learning not a lecture and not a class room based taught programme alone. You are here to learn and learn well.

**Explore the Android Studio**

Diagram 1 - How to start Android Studio

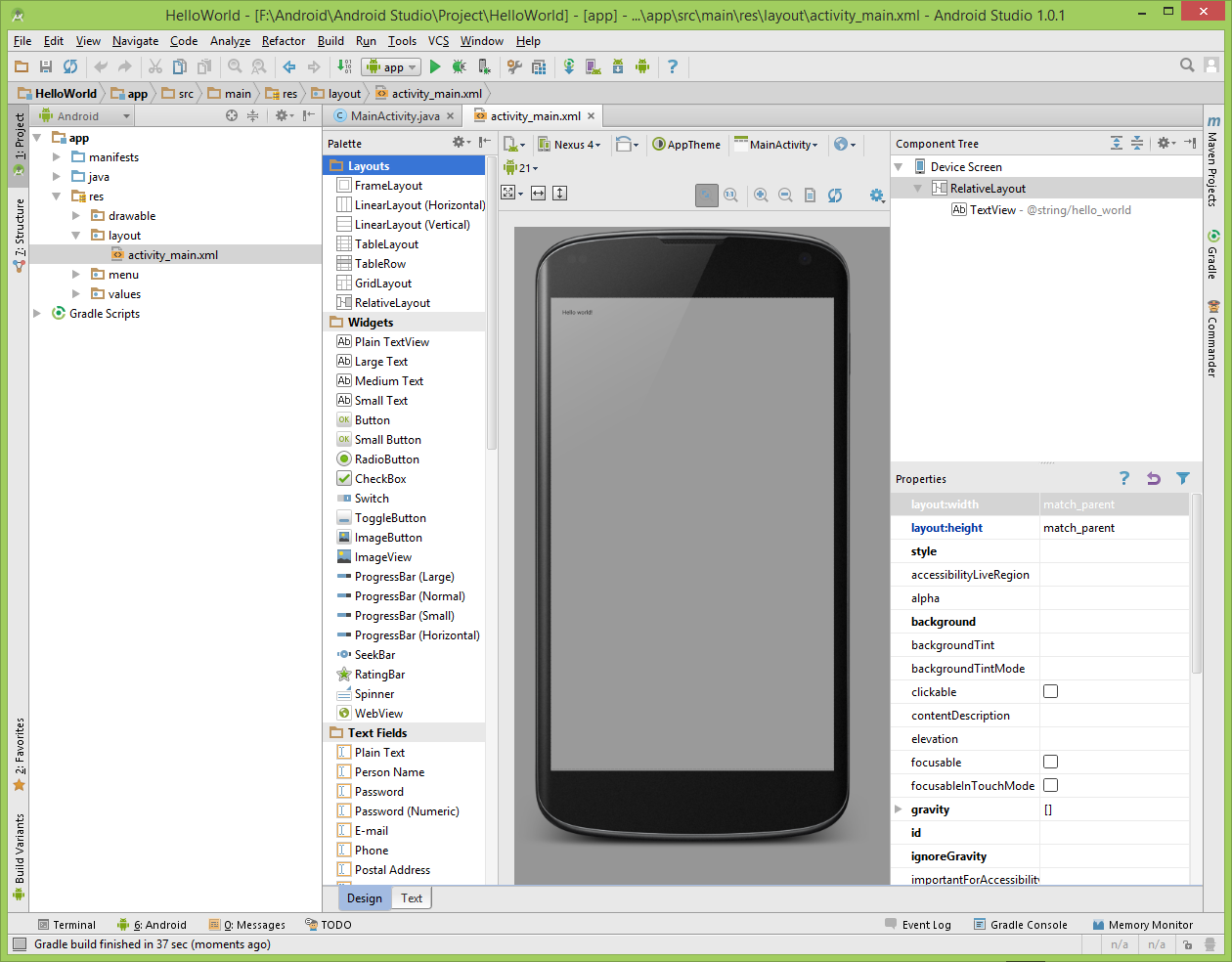
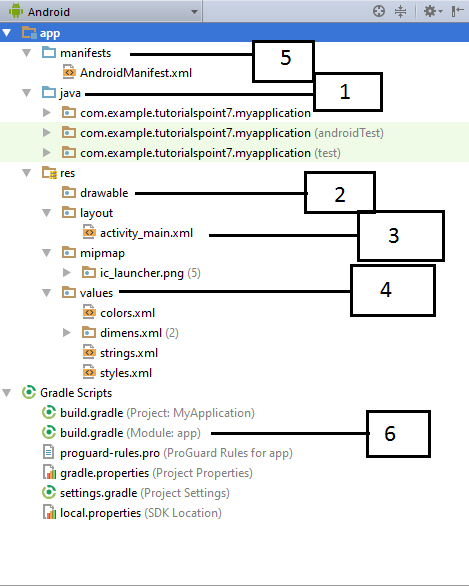
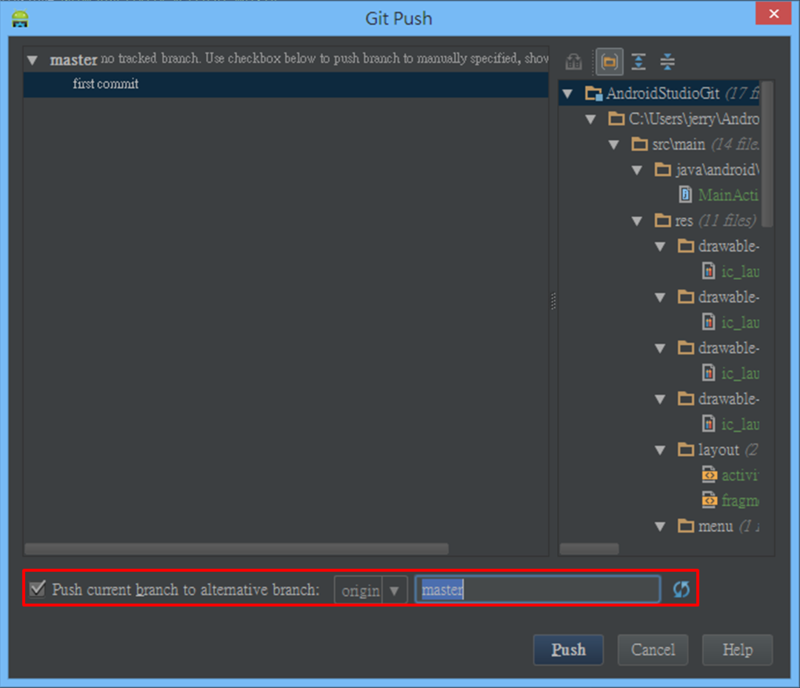


Diagram 2



**Android Studio and GitHub or Bitbucket**

Diagram 1- Attach Github or bitbucket to the Android Studio



**Here is how I did it without plug-ins:**

1- Create the **repository** on your Bitbucket account

2- **Create your project** in Android Studio

3- In Android Studio, **Go to VCS**

4- Choose 'Enable version control'

5- Choose Git and press OK

6- **Right click** on your project, choose Git then click Add

7- Open Terminal in Android studio

8- Go to your Bitbucket repository Overview

9- Click on 'I have an existing Project'

10- Copy the 'git remote add origin ... etc' line to your terminal and **press enter**

11- Click on 'Commit Changes', write your comment then press Commit and push

References

* https://bitbucket.org/variabletech/libnode-android-public/wiki/Project%20Setup%20(Android%20Studio)
* http://stackoverflow.com/questions/19099244/how-to-import-a-project-into-bitbucket-repository-from-android-studio

**App Design in Android and The Wireframe and/** or Storyboarding third party open source tools

Diagram 1 - MVC Design Pattern

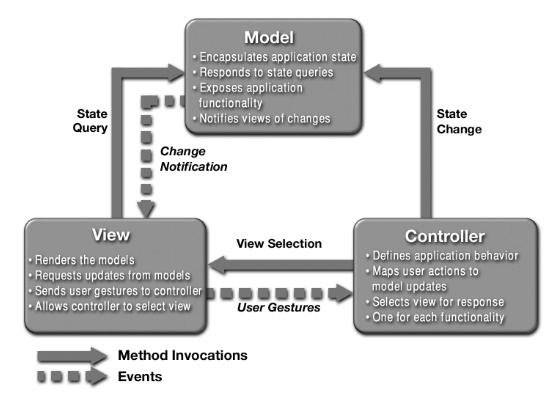


Diagram 2 - MVC Design Pattern

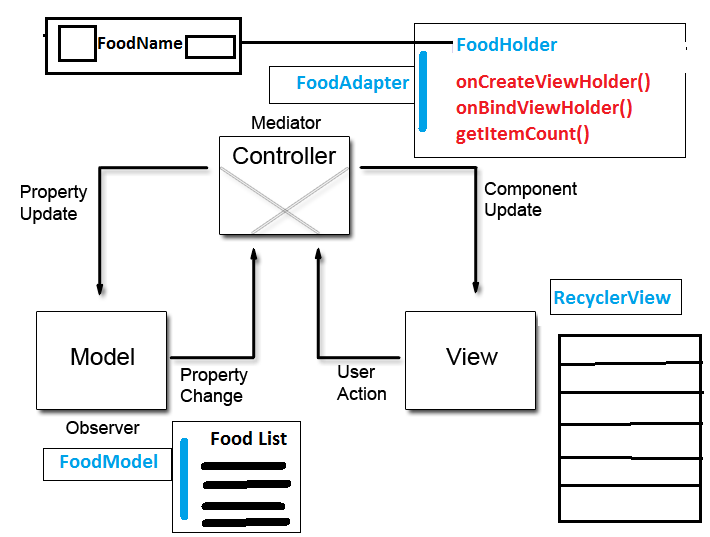


Diagram - 3 - Dependency Injection (DI) Design Pattern in Android

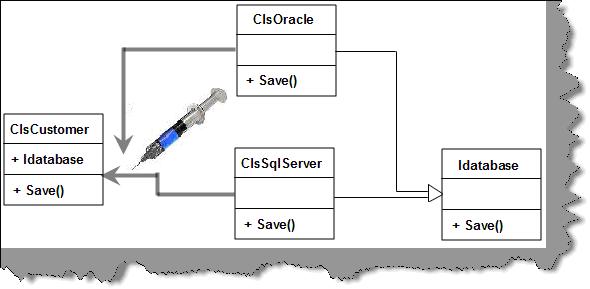


Diagram 4 - Consumer - Producer Design Pattern in Android

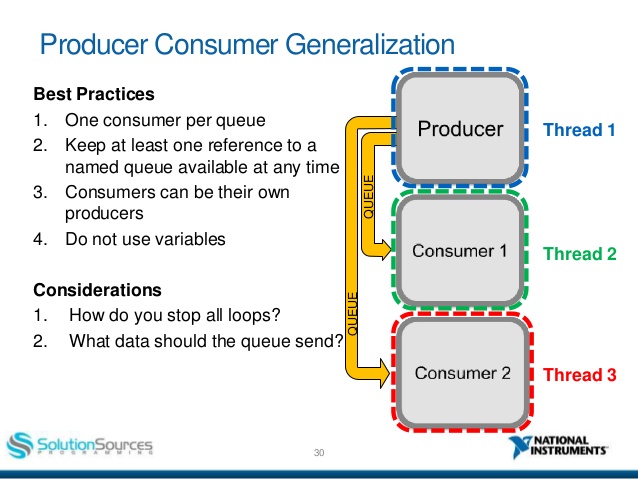


Diagram 5 - FRP Design Pattern in Android e.g. Observer Design Pattern



Diagram 6 - Adapter Design Pattern in Android

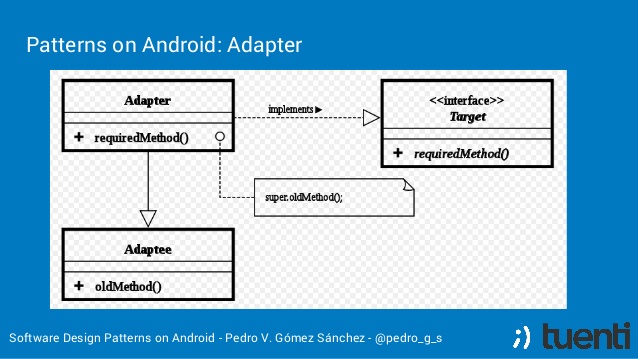


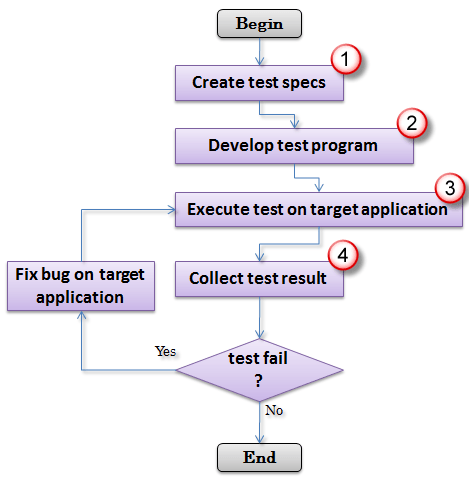
Diagram 7 - Creational Design Pattern in Android



So many design patterns are in use in Android App Development right now each is very useful and the more familiar the developer and designer are to the design pattern workings the more these will be used. Solution Architects, technical architects, mobility architects, sr. developers use these approaches to app design.

**The Android Testing Framework**

Diagram 1 - This could be any kind of testing in testing phase.



This could be Unit / Mock / Functional testing in development phase or in testing phase.

Diagram 2 - The image that should happen in your mind when Testing is mentioned in Android

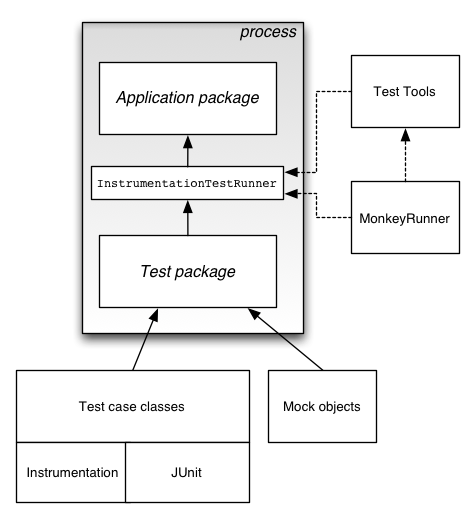
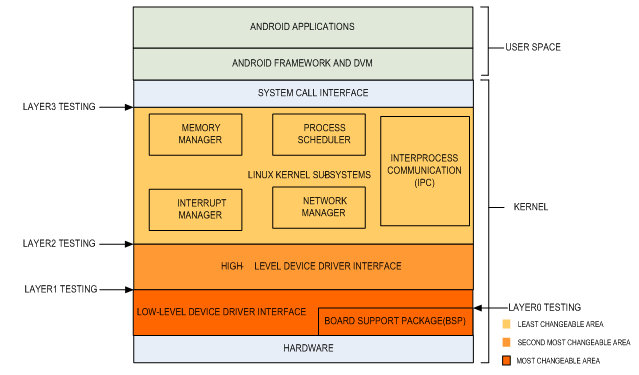


Diagram 3 - Another perspective of Android Testing



**The Android Testing Framework**

Diagram 1 - A Poster to help you check everything

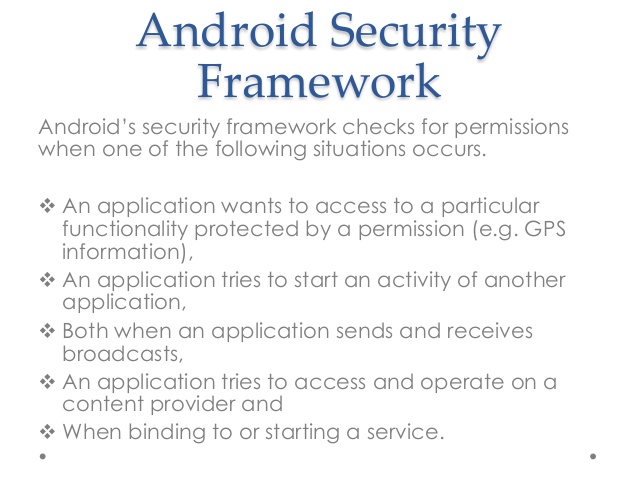


Diagram 2 - Revisit the Android Framework to understand Android Security Framework

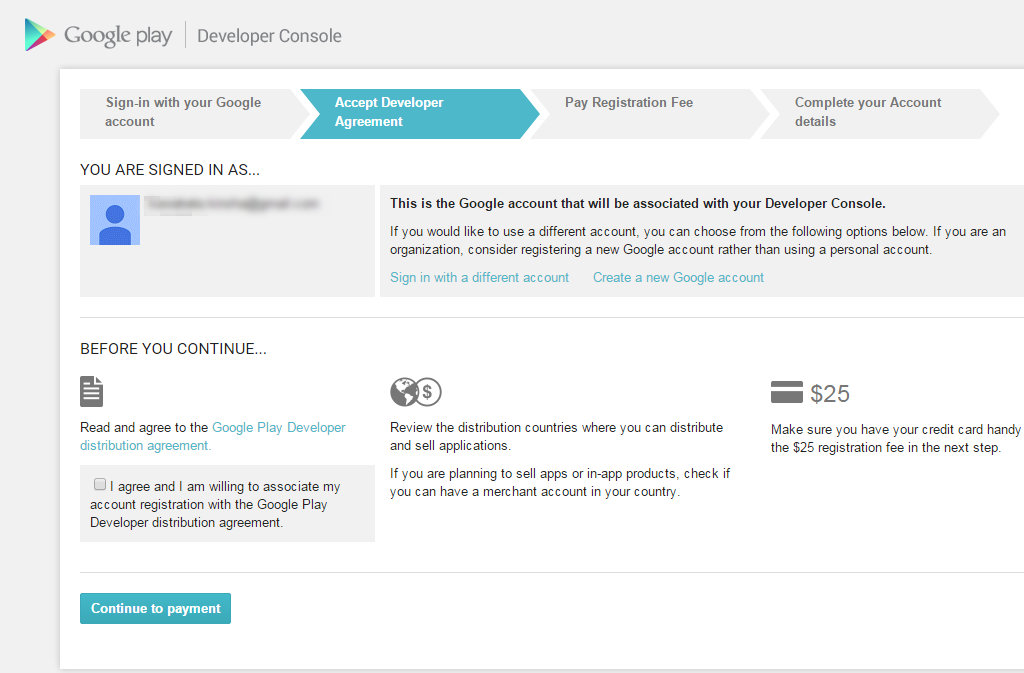


**Packaging, Deploying and Distributing or Selling Your App**

Diagram 1 -



Diagram 2 - sample images to help learner visualize with ease items like Developer Console



**Topics to learn**

* Android as an operating system, Open Handset Alliance
* Android architecture, the Linux kernel and other layers
* Lifecycle of an Android App ...requirements, design, development, testing, packaging, distribution
* Understanding the ***Anatomy of an Android App***
* The Android **fragmentation issue** and its solutions
  + The diverse Android devices in smart phones, wearables, tv, fridge, auto, glass, etc.
  + The diverse Android versions from Gingerbread to Noughat
  + The presence of huge number of bugs in older version fixed in newer version
  + The variety of vendors like Samsung, LG, HTC and the issues they bring ordinary folk
* **Setting Up** Your Android Development Environment
  + Download and install JDK and Android Studio
  + Setting Up for device debugging
  + Explore the Android SDK
    - The Android SDK License Agreement
    - Important packages in the Android SDK
    - Third-party Android APIs
  + Explore the Android Studio, Gradle and other parts
  + Preview Android samples and using SDK samples
  + Attach Github or bitbucket to the Android Studio
* App Design in Android and The Wireframe and/ or Storyboarding third party open source tools
  + WireframeSketcher **tool** with **Android**
  + Fluid UI
* The Android **Testing** Framework
  + A test-driven development approach in Android
  + Set up a Android Virtual Device for App Testing
  + Testing on a huge range of devices with cloud-based testing
  + Create and use a test project
  + Debugging with Log.d and LogCat
  + Use of DDMS, debugging an Android app, ADB, TraceView, Lint, Hierarchy Viewer
  + Writing automated tests - performance
* The Android **Security** framework
  + Security and Permissions
  + Use of Encryption in Android
  + Android Security Concepts -
    - Signatures and Keys - how to generate a new, unique key for your app
    - Android Permissions - types (5) five protection levels
    - Custom Permissions - define your own permissions
    - Protecting User Data
    - Verifying Calling Apps
    - Client Side Data Encryption - Android Crypto API for encrypting and decrypting data in Android - javax.crypto
    - Android Key Chain Management
    - Device Management API
* Packaging, Deploying and **Distributing** or Selling Your App
  + Introduction to the final stages of App development
  + Anatomy of an Android app - the APK file
  + Pre-Release stages
  + Packaging
  + Using Android Play or Google Play Store
  + Using AdMob and ProGuard
  + Using a Continuous Integration System and the Github

Illustrated Example

* https://www.tutorialspoint.com/android/android\_overview.htm
* https://code.tutsplus.com/articles/android-from-scratch-an-overview-of-android-application-development--cms-25972
* https://developer.android.com/work/overview.html
* https://developer.android.com/guide/index.html
* https://developer.android.com/guide/components/fundamentals.html
* http://www.openhandsetalliance.com/android\_overview.html

Chapter Two **Main Building Blocks of Android**

Objective There are four main building blocks + the Android Manifest file and the App itself.

Background

### [The out-of-memory killer](http://www.vogella.com/tutorials/AndroidLifeCycle/article.html#the-out-of-memory-killer)

In an ideal case all Android applications started by the user remain in memory. That makes applications switching much faster for the user. But in reality the available memory on an Android device is limited. To manage these limited resources the Android system can terminate running processes. If the Android system needs to free up resources it follows a simple set of rules. Every process **gets a priority** in the following order.

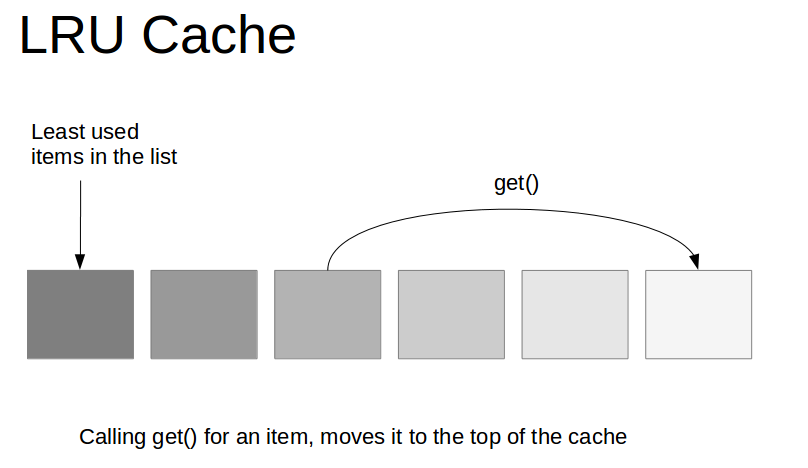
If the Android system needs to terminate processes it follows the following priority system.

| *Table 1. Priorities* | | |
| --- | --- | --- |
| **Process status** | **Description** | **Priority** |
| Foreground | An application in which the user is interacting with an activity, or which has an service which is bound to such an activity. Also if a service is executing one of its lifecycle methods or a broadcast receiver which runs its onReceive() method. | 1 |
| Visible | User is not interacting with the activity, but the activity is still (partially) visible or the application has a service which is used by a inactive but visible activity. | 2 |
| Service | Application with a running service which does not qualify for 1 or 2. | 3 |
| Background | Application with only stopped activities and without a service or executing receiver. Android keeps them in a least recent used (LRU) list and if requires terminates the one which was least used. | 4 |
| Empty | Application without any active components. | 5 |

All processes in the empty list are added to **a least recently used list** (LRU list).

The processes which are **at the beginning of this lists** will be the ones killed by the out-of-memory killer. If an application is restarted by the user, its gets moved to the end of this queue.

If it reaches **the lowest priority again**, as indicated by the following graphic.



### [Application](http://www.vogella.com/tutorials/AndroidLifeCycle/article.html#application)

The **application object** is created before any of your Android components are started. If you do not specify one in your **AndroidManifest.xml** file, the Android system creates a default object for you.

It is started in a new process with **a unique ID** under a unique user.

This object provides the following main life-cycle methods:

* **onCreate()** - called before the first components of the application starts
* **onLowMemory()** - called when the Android system requests that the application cleans up memory
* **onTrimMemory()** - called when the Android system requests that the application cleans up memory. This message includes an indicator in which position the application is. For example the constant TRIM\_MEMORY\_MODERATE indicates that the process is around the middle of the background LRU list; freeing memory can help the system keep other processes running later in the list for better overall performance.
* **onTerminate() -** only for testing, not called in production
* onConfigurationChanged() - called whenever the configuration changes

The **application object starts before** any component and runs at least as long as another component of the application runs.

**Every application** must have an **AndroidManifest.xml** file (with precisely that name) in its root directory. **The manifest file** provides essential information about your app to the Android system, which the system must have before it can run any of the app's code.

Among other things, the manifest file does the following:

* It names the Java package for the application. The package name serves as a unique identifier for the application.
* It describes the components of the application, which include the activities, services, broadcast receivers, and content providers that compose the application. It also names the classes that implement each of the components and publishes their capabilities, such as the [Intent](https://developer.android.com/reference/android/content/Intent.html) messages that they can handle. These declarations inform the Android system of the components and the conditions in which they can be launched.
* It determines the processes that host the application components.
* It declares the **permissions that the application** must have in order to access protected parts of the API and interact with other applications. It also declares the permissions that others are required to have in order to interact with the application's components.
* It lists the **Instrumentation** classes that provide profiling and other information as the application runs. These declarations are present in the manifest only while the application is being developed and are removed before the application is published.
* It declares the minimum level of the Android API that the application requires.
* It lists the libraries that the application must be linked against.

## Manifest file structure

The code snippet below shows the general structure of the manifest file and every element that it can contain. Each element, along with all of its attributes, is fully documented in a separate file.

**Tip**: To view detailed information about any of the elements that are mentioned within the text of this document, simply click the element name.

Here is an example of the manifest file:

<?xml version="1.0" encoding="utf-8"?>  
  
[<manifest>](https://developer.android.com/guide/topics/manifest/manifest-element.html)  
  
    [**<uses-permission />**](https://developer.android.com/guide/topics/manifest/uses-permission-element.html)[**<permission />**](https://developer.android.com/guide/topics/manifest/permission-element.html)[**<permission-tree />**](https://developer.android.com/guide/topics/manifest/permission-tree-element.html)[**<permission-group />**](https://developer.android.com/guide/topics/manifest/permission-group-element.html)[**<instrumentation />**](https://developer.android.com/guide/topics/manifest/instrumentation-element.html)[**<uses-sdk />**](https://developer.android.com/guide/topics/manifest/uses-sdk-element.html)[**<uses-configuration />**](https://developer.android.com/guide/topics/manifest/uses-configuration-element.html)[**<uses-feature />**](https://developer.android.com/guide/topics/manifest/uses-feature-element.html)[**<supports-screens />**](https://developer.android.com/guide/topics/manifest/supports-screens-element.html)[**<compatible-screens />**](https://developer.android.com/guide/topics/manifest/compatible-screens-element.html)[**<supports-gl-texture />**](https://developer.android.com/guide/topics/manifest/supports-gl-texture-element.html)    
  
    [<application>](https://developer.android.com/guide/topics/manifest/application-element.html)  
        [<activity>](https://developer.android.com/guide/topics/manifest/activity-element.html)  
            [<intent-filter>](https://developer.android.com/guide/topics/manifest/intent-filter-element.html)  
                [<action />](https://developer.android.com/guide/topics/manifest/action-element.html)  
                [<category />](https://developer.android.com/guide/topics/manifest/category-element.html)  
                [<data />](https://developer.android.com/guide/topics/manifest/data-element.html)  
            [</intent-filter>](https://developer.android.com/guide/topics/manifest/intent-filter-element.html)  
            [<meta-data />](https://developer.android.com/guide/topics/manifest/meta-data-element.html)  
        [</activity>](https://developer.android.com/guide/topics/manifest/activity-element.html)  
  
        [<activity-alias>](https://developer.android.com/guide/topics/manifest/activity-alias-element.html)  
            [<intent-filter>](https://developer.android.com/guide/topics/manifest/intent-filter-element.html) . . . [</intent-filter>](https://developer.android.com/guide/topics/manifest/intent-filter-element.html)  
            [<meta-data />](https://developer.android.com/guide/topics/manifest/meta-data-element.html)  
        [</activity-alias>](https://developer.android.com/guide/topics/manifest/activity-alias-element.html)  
  
        [<service>](https://developer.android.com/guide/topics/manifest/service-element.html)  
            [<intent-filter>](https://developer.android.com/guide/topics/manifest/intent-filter-element.html) . . . [</intent-filter>](https://developer.android.com/guide/topics/manifest/intent-filter-element.html)  
            [<meta-data/>](https://developer.android.com/guide/topics/manifest/meta-data-element.html)  
        [</service>](https://developer.android.com/guide/topics/manifest/service-element.html)  
  
        [<receiver>](https://developer.android.com/guide/topics/manifest/receiver-element.html)  
            [<intent-filter>](https://developer.android.com/guide/topics/manifest/intent-filter-element.html) . . . [</intent-filter>](https://developer.android.com/guide/topics/manifest/intent-filter-element.html)  
            [<meta-data />](https://developer.android.com/guide/topics/manifest/meta-data-element.html)  
        [</receiver>](https://developer.android.com/guide/topics/manifest/receiver-element.html)  
  
        [<provider>](https://developer.android.com/guide/topics/manifest/provider-element.html)  
            [<grant-uri-permission />](https://developer.android.com/guide/topics/manifest/grant-uri-permission-element.html)  
            [<meta-data />](https://developer.android.com/guide/topics/manifest/meta-data-element.html)  
            [<path-permission />](https://developer.android.com/guide/topics/manifest/path-permission-element.html)  
        [</provider>](https://developer.android.com/guide/topics/manifest/provider-element.html)  
  
        [<uses-library />](https://developer.android.com/guide/topics/manifest/uses-library-element.html)  
    [</application>](https://developer.android.com/guide/topics/manifest/application-element.html)  
[</manifest>](https://developer.android.com/guide/topics/manifest/manifest-element.html)

The following list contains all of the elements that can appear in the manifest file, in alphabetical order:

* [<action>](https://developer.android.com/guide/topics/manifest/action-element.html)
* [<activity>](https://developer.android.com/guide/topics/manifest/activity-element.html)
* [<activity-alias>](https://developer.android.com/guide/topics/manifest/activity-alias-element.html)
* [<application>](https://developer.android.com/guide/topics/manifest/application-element.html)
* [<category>](https://developer.android.com/guide/topics/manifest/category-element.html)
* [<data>](https://developer.android.com/guide/topics/manifest/data-element.html)
* [<grant-uri-permission>](https://developer.android.com/guide/topics/manifest/grant-uri-permission-element.html)
* [<instrumentation>](https://developer.android.com/guide/topics/manifest/instrumentation-element.html)
* [<intent-filter>](https://developer.android.com/guide/topics/manifest/intent-filter-element.html)
* [<manifest>](https://developer.android.com/guide/topics/manifest/manifest-element.html)
* [<meta-data>](https://developer.android.com/guide/topics/manifest/meta-data-element.html)
* [<permission>](https://developer.android.com/guide/topics/manifest/permission-element.html)
* [<permission-group>](https://developer.android.com/guide/topics/manifest/permission-group-element.html)
* [<permission-tree>](https://developer.android.com/guide/topics/manifest/permission-tree-element.html)
* [<provider>](https://developer.android.com/guide/topics/manifest/provider-element.html)
* [<receiver>](https://developer.android.com/guide/topics/manifest/receiver-element.html)
* [<service>](https://developer.android.com/guide/topics/manifest/service-element.html)
* [<supports-screens>](https://developer.android.com/guide/topics/manifest/supports-screens-element.html)
* [<uses-configuration>](https://developer.android.com/guide/topics/manifest/uses-configuration-element.html)
* [<uses-feature>](https://developer.android.com/guide/topics/manifest/uses-feature-element.html)
* [<uses-library>](https://developer.android.com/guide/topics/manifest/uses-library-element.html)
* [<uses-permission>](https://developer.android.com/guide/topics/manifest/uses-permission-element.html)
* [<uses-sdk>](https://developer.android.com/guide/topics/manifest/uses-sdk-element.html)

**Note**: These are the only legal elements – you cannot add your own elements or attributes.

## File conventions

This section describes the conventions and rules that apply generally to all of the elements and attributes in the manifest file.

**Elements**

Only the [<manifest>](https://developer.android.com/guide/topics/manifest/manifest-element.html) and [<application>](https://developer.android.com/guide/topics/manifest/application-element.html) elements are required. They each must be present and can occur only once. Most of the other elements can occur many times or not at all. However, at least some of them must be present before the manifest file becomes useful.

If an element contains anything at all, it contains other elements. All of the values are set through attributes, not as character data within an element.

Elements at the same level are generally **not** ordered. For example, the [<activity>](https://developer.android.com/guide/topics/manifest/activity-element.html), [<provider>](https://developer.android.com/guide/topics/manifest/provider-element.html), and [<service>](https://developer.android.com/guide/topics/manifest/service-element.html) elements can be intermixed in any sequence. There are two key exceptions to this rule:

* An [<activity-alias>](https://developer.android.com/guide/topics/manifest/activity-alias-element.html) element must follow the [<activity>](https://developer.android.com/guide/topics/manifest/activity-element.html) for which it is an alias.
* The [<application>](https://developer.android.com/guide/topics/manifest/application-element.html) element must be the last element inside the [<manifest>](https://developer.android.com/guide/topics/manifest/manifest-element.html) element. In other words, the </application> closing tag must appear immediately before the </manifest> closing tag.

**Attributes**

In a formal sense, all attributes are optional. However, there are some attributes that must be specified so that an element can accomplish its purpose. Use the documentation as a guide.

For truly optional attributes, it mentions a default value or states what happens in the absence of a specification.

Except for some attributes of the root [<manifest>](https://developer.android.com/guide/topics/manifest/manifest-element.html) element, all attribute names begin with an android: prefix. For example, android:alwaysRetainTaskState. Because the prefix is universal, the documentation generally omits it when referring to attributes by name.

**Declaring class names**

Many elements correspond to Java objects, including elements for the application itself (the [<application>](https://developer.android.com/guide/topics/manifest/application-element.html) element) and its principal components: activities ([<activity>](https://developer.android.com/guide/topics/manifest/activity-element.html)), services ([<service>](https://developer.android.com/guide/topics/manifest/service-element.html)), broadcast receivers ([<receiver>](https://developer.android.com/guide/topics/manifest/receiver-element.html)), and content providers ([<provider>](https://developer.android.com/guide/topics/manifest/provider-element.html)).

If you define a subclass, as you almost always would for the component classes ([Activity](https://developer.android.com/reference/android/app/Activity.html), [Service](https://developer.android.com/reference/android/app/Service.html), [BroadcastReceiver](https://developer.android.com/reference/android/content/BroadcastReceiver.html), and [ContentProvider](https://developer.android.com/reference/android/content/ContentProvider.html)), the subclass is declared through a name attribute. The name must include the full package designation.

For example, a [Service](https://developer.android.com/reference/android/app/Service.html) subclass might be declared as follows:

<manifest . . . >  
    <application . . . >  
        <service android:name="com.example.project.SecretService" . . . >  
            . . .  
        </service>  
        . . .  
    </application>  
</manifest>

However, if the first character of the string is a period, the application's package name (as specified by the [<manifest>](https://developer.android.com/guide/topics/manifest/manifest-element.html) element's [package](https://developer.android.com/guide/topics/manifest/manifest-element.html#package)attribute) is appended to the string. The following assignment is the same as that shown above:

<manifest package="com.example.project" . . . >  
    <application . . . >  
        <service android:name=".SecretService" . . . >  
            . . .  
        </service>  
        . . .  
    </application>  
</manifest>

When starting a component, the Android system creates an instance of the named subclass. If a subclass isn't specified, it creates an instance of the base class.

**Multiple values**

If more than one value can be specified, the element is almost always repeated, rather than multiple values being listed within a single element.

For example, an intent filter can list several actions:

<intent-filter . . . >  
    <action android:name="android.intent.action.EDIT" />  
    <action android:name="android.intent.action.INSERT" />  
    <action android:name="android.intent.action.DELETE" />  
    . . .  
</intent-filter>

**Resource values**

Some attributes have values that can be displayed to users, such as a label and an icon for an activity. The values of these attributes should be localized and set from a resource or theme.

Resource values are expressed in the following format:

@[*package*:]*type*/*name*

You can omit the *package* name if the resource is in the same package as the application. The *type* is a type of resource, such as string or drawable, and the *name* is the name that identifies the specific resource.

Here is an example:

<activity android:icon="@drawable/smallPic" . . . >

The values from a theme are expressed similarly, but with an initial ? instead of @:

?[*package*:]*type*/*name*

**String values**

Where an **attribute value** is a string, you must use double backslashes (\\) to escape characters, such as \\n for a newline or \\uxxxx for a Unicode character.

## File features

The following sections describe the way that some Android features are reflected in the manifest file.

### Intent filters

The core components of an application, such as its activities, services, and broadcast receivers, are activated by *intents*. An intent is a bundle of information (an [Intent](https://developer.android.com/reference/android/content/Intent.html) object) describing a desired action, including the data to be acted upon, the category of component that should perform the action, and other pertinent instructions.

The Android system locates an appropriate component that can **respond** to the intent, launches a **new instance** of the component if one is needed, and passes it the [Intent](https://developer.android.com/reference/android/content/Intent.html) object.

The components advertise the types of intents that they can respond to through *intent filters*. Since the Android system must learn the intents that a component can handle before it launches the component, intent filters are specified in the manifest as [<intent-filter>](https://developer.android.com/guide/topics/manifest/intent-filter-element.html) elements.

A component can have any number of filters, each one describing a different capability.

An intent that explicitly names a target component activates that component, so the filter doesn't play a role. An intent that doesn't specify a target by name can activate a component only if it can pass through one of the component's filters.

For information about how [Intent](https://developer.android.com/reference/android/content/Intent.html) objects are tested against intent filters, see the [Intents and Intent Filters](https://developer.android.com/guide/components/intents-filters.html) document.

### Icons and labels

A number of elements have icon and label attributes for a small icon and a text label that can be displayed to users.

Some also have a description attribute for longer, explanatory text that can also be shown on-screen.

For example, the [<permission>](https://developer.android.com/guide/topics/manifest/permission-element.html) element has all three of these attributes so that when the user is asked whether to grant the permission to an application that has requested it, an icon representing the permission, the name of the permission, and a description of what it entails are all presented to the user.

In every case, the icon and label that are set in a containing element become the default icon and label settings for all of the container's sub-elements. Thus, the icon and label that are set in the [<application>](https://developer.android.com/guide/topics/manifest/application-element.html) element are the default icon and label for each of the application's components.

Similarly, the icon and label that are set for a component, such as an [<activity>](https://developer.android.com/guide/topics/manifest/activity-element.html) element, are the default settings for each of the component's [<intent-filter>](https://developer.android.com/guide/topics/manifest/intent-filter-element.html)elements.

If an [<application>](https://developer.android.com/guide/topics/manifest/application-element.html) element sets a label, but an activity and its intent filter do not, the application label is treated as the label for both the activity and the intent filter.

The icon and label that are set for an intent filter represent a component whenever the component is presented to the user and fulfils the function that is advertised by the filter.

For example, a filter with android.intent.action.MAIN and android.intent.category.LAUNCHER settings advertises an activity as one that initiates an application. That is, as one that should be displayed in the application launcher. The icon and label that are set in the filter are displayed in the launcher.

### Permissions

A *permission* is a restriction that limits access to a part of the code or to data on the device. The limitation is imposed to protect critical data and code that could be misused to distort or damage the user experience.

Each permission is identified by a unique label. Often the label indicates the action that's restricted.

Here are some **permissions** that are defined by Android:

* android.permission.CALL\_EMERGENCY\_NUMBERS
* android.permission.READ\_OWNER\_DATA
* android.permission.SET\_WALLPAPER
* android.permission.DEVICE\_POWER

A **feature can be protected** by only one permission.

If an application needs access to a feature that is protected by a permission, it must declare that it requires the permission with **a**[**<uses-permission>**](https://developer.android.com/guide/topics/manifest/uses-permission-element.html)**element** in the manifest. When the application is installed on the device, the installer determines whether to grant the requested permission by checking the authorities that signed the application's certificates and, in some cases, asking the user. If the permission is granted, the application is able to use the protected features. If not, its attempts to access those features fail without any notification to the user.

An application can also protect its own components with permissions.

It can employ any of the permissions that are defined by Android, as listed **in android.Manifest.permission**, or declared by other applications. It can also define its own. A new permission is declared with the **<permission>**element.

For example, **an activity could be protected** as follows:

<manifest . . . >  
    <permission android:name="com.example.project.DEBIT\_ACCT" . . . />  
    <uses-permission android:name="com.example.project.DEBIT\_ACCT" />  
    . . .  
    <application . . .>  
        <activity android:name="com.example.project.FreneticActivity"  
                  android:permission="com.example.project.DEBIT\_ACCT"  
                  . . . >  
            . . .  
        </activity>  
    </application>  
</manifest>

Note that, in this example, the DEBIT\_ACCT permission is not only declared with the [<permission>](https://developer.android.com/guide/topics/manifest/permission-element.html) element, its use is also requested with the [<uses-permission>](https://developer.android.com/guide/topics/manifest/uses-permission-element.html) element. You must request its use in order for other components of the application to launch the protected activity, even though the protection is imposed by the application itself.

If, in the same example shown above, the permission attribute was set to a permission that is declared elsewhere, such as android.permission.CALL\_EMERGENCY\_NUMBERS, it would not be necessary to declare it again with a [<permission>](https://developer.android.com/guide/topics/manifest/permission-element.html) element. However, it would still be necessary to request its use with [<uses-permission>](https://developer.android.com/guide/topics/manifest/uses-permission-element.html).

The [<permission-tree>](https://developer.android.com/guide/topics/manifest/permission-tree-element.html) element declares a namespace for a group of permissions that are defined in code, and the [<permission-group>](https://developer.android.com/guide/topics/manifest/permission-group-element.html) defines a label for a set of permissions, both those declared in the manifest with [<permission>](https://developer.android.com/guide/topics/manifest/permission-element.html) elements and those declared elsewhere. This affects only how the permissions are grouped when presented to the user.

The [**<permission-group>**](https://developer.android.com/guide/topics/manifest/permission-group-element.html)**element** does not specify the permissions that belong to the group, but it gives the group a name. You can place a permission in the group by assigning the group name to the [<permission>](https://developer.android.com/guide/topics/manifest/permission-element.html) element's **permissionGroup** attribute.

### Libraries

Every application is linked against the default Android library, which includes the basic packages for building applications (with common classes such as Activity, Service, Intent, View, Button, Application, and ContentProvider).

However, some packages reside in their own libraries.

If your application uses code from any of these packages, it must explicitly ask to be linked against them.

The manifest must contain a separate [<uses-library>](https://developer.android.com/guide/topics/manifest/uses-library-element.html) element to name each of the libraries. You can find the library name in the documentation for the package.

The **AndroidManifest.xml file** *contains information of your package*, including components of the application such as activities, services, broadcast receivers, content providers etc.

It performs some other tasks also:

* It is **responsible to protect the application** to access any protected parts by providing the permissions.
* It also **declares the android api** that the application is going to use.
* It **lists the instrumentation classes**. The instrumentation classes provides profiling and other informations. These informations are removed just before the application is published etc.

This is the required xml file for all the android application and located inside the root directory.

Topics to learn

* + - * 1. About the App and what happens in the beginning before any component launches
        2. The Android Manifest file

Declaring an activity and a service and other components

The elements that make up a manifest file

Illustrated Example

* https://developer.android.com/guide/topics/manifest/manifest-intro.html
* https://www.javatpoint.com/AndroidManifest-xml-file-in-android
* http://javapapers.com/android/android-manifest/
* http://simpledeveloper.com/android-application-manifest-file/
* https://code.tutsplus.com/tutorials/android-sdk-project-manifest--mobile-20606
* http://www.vogella.com/tutorials/AndroidLifeCycle/article.html#the-out-of-memory-killer
* https://www.raywenderlich.com/116580/introduction-to-android-activities-tutorial
* http://tutorials.jenkov.com/android/activity.html
* http://programmerguru.com/android-tutorial/android-activity/
* https://www.javatpoint.com/android-life-cycle-of-activity
* https://www.mkyong.com/android/android-activity-from-one-screen-to-another-screen/

Chapter Two ...Contd. **Main Building Blocks of Android**

Objective There are four main building blocks + the Android Manifest file and the App itself.

Topics to learn

* + - * 1. **Activities**, Creating a project and an activity, activity lifecycle

Understanding the activity life cycle, onStart() and other methods

Saving and restoring the activity information

Understanding tasks and the back stack

Handling configuration changes for diverse Android devices

Using Fragment - display multiple fragments at once and using dialog fragments

Add, remove and replace fragments

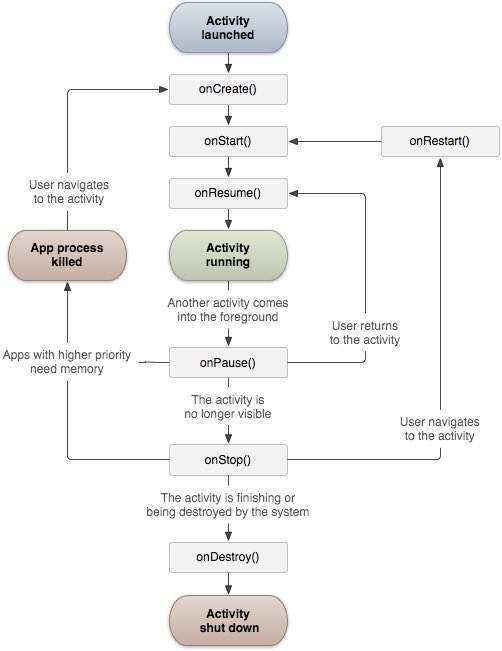
Finding Activity using fragment

Specialized Fragments - ListFragment, DialogFragment, PreferenceFragment

Multiple Activities

Background

An activity represents a single screen with a user interface just like window or frame of Java.Android activity is the subclass of ContextThemeWrapper class.



**Activities** are one of the fundamental building blocks of apps on the Android platform.

They serve as the entry point for a user's interaction with an app, and are also central to how a user navigates within an app (as with the Back button) or between apps (as with the Recents button).

Skilfully managing activities allows you to ensure that, for example:

* Orientation changes take place smoothly without disrupting the user experience.
* User data is not lost during activity transitions.
* The system kills processes when it's appropriate to do so.

The Graphical User Interface the user sees on launching an app - example WhatsApp, SnapChat, Instagram, Facebook, Twitter, Yumist, etc. are all the first Activity started using the **onCreate()** method.

The Activity has methods onCreate() that begins the story of an Activity, it has onPause() so that if the user is on WhatsApp for example and there is an incoming call the UI is paused and the specific UI changes priority from ***foreground*** to ***background*** and it is put on the backstack. After the incoming call is over and ceased the priority is again reversed and the UI is shown to the user without a lag or a pause.

Activity has other methods like onRestart(), onStop(), onDestroy() and so on. The use of these methods make up the Activity Life cycle you saw in previous page diagram.

Multiple Activities are seen in any Android App - check Google Play Store or Android Police or any play store for Android. Each app is illustrated by its UI Screens and each screen represents an Activity. There could be approximately 5 to 7 activities.

From the first Activity it is possible to call a second Activity - example in WhatsApp there are a few different UI screens and a few associated Activities that have to be called whenever the user wishes to use these UI screens or Activities.

Can you tell what are the different Activities in WhatsApp app?

Illustrated Example

* https://www.tutorialspoint.com/android/android\_acitivities.htm
* https://developer.android.com/guide/components/activities/index.html
* https://developer.android.com/training/basics/firstapp/starting-activity.html
* http://www.vogella.com/tutorials/AndroidLifeCycle/article.html

Background

* [Starting other Android components via intents](http://www.vogella.com/tutorials/AndroidIntent/article.html#starting-other-android-components-via-intents)
* [**Data transfer** between activities](http://www.vogella.com/tutorials/AndroidIntent/article.html#data-transfer-between-activities)
* [Registering for intents via intent filters](http://www.vogella.com/tutorials/AndroidIntent/article.html#intentfilter)
* [Activity communication with intents](http://www.vogella.com/tutorials/AndroidIntent/article.html#exercise-activity-communication-with-intents)
* [**Using the share intent**](http://www.vogella.com/tutorials/AndroidIntent/article.html#exercise-using-the-share-intent)
* [Register an activity as browser](http://www.vogella.com/tutorials/AndroidIntent/article.html#exercise-register-an-activity-as-browser)
* [Picking an image via an intent](http://www.vogella.com/tutorials/AndroidIntent/article.html#exercise-picking-an-image-via-an-intent)
* [Using different implicit intents](http://www.vogella.com/tutorials/AndroidIntent/article.html#exercise-using-different-implicit-intents)

## [Starting other Android components via intents](http://www.vogella.com/tutorials/AndroidIntent/article.html#starting-other-android-components-via-intents)

### [What are intents?](http://www.vogella.com/tutorials/AndroidIntent/article.html#what-are-intents)

Android application components can connect to other Android applications. This connection is based on a task description represented by an Intentobject.

**Intents a**re asynchronous messages which allow application components to request functionality from other Android components. Intents allow you to interact with components from the same applications as well as with components contributed by other applications. For example, an activity can start an external activity for taking a picture.

**Intents** are objects of the **android.content.Intent** type. Your code can send them to the Android system defining the components you are targeting. For example, via the startActivity() method you can define that the intent should be used to start an activity.

An intent can contain data via a Bundle. This data can be used by the receiving component.

In Android the reuse of other application components is a concept known as task. An application can access other Android components to achieve a task. For example, from a component of your application you can trigger another component in the Android system, which manages photos, even if this component is not part of your application. In this component you select a photo and return to your application to use the selected photo.

### [Starting activities or services](http://www.vogella.com/tutorials/AndroidIntent/article.html#starting-activities-or-services)

To start an activity, use the method startActivity(intent). This method is defined on the Context object which Activity extends.

The following code demonstrates how you can start another activity via an intent.

# Start the activity connect to the

# specified **class**

**Intent** i = **new** Intent(this, ActivityTwo.class);

startActivity(i);

Activities which are started by other Android activities are called sub-activities. This wording makes it easier to describe which activity is meant.

To start a services via intents, use the startService(Intent) method call.

### [Sending out explicit or implicit intents](http://www.vogella.com/tutorials/AndroidIntent/article.html#sending-out-explicit-or-implicit-intents)

Android supports explicit and implicit intents. An application can define the target component directly in the intent (**explicit intent)**

or

ask the Android system to evaluate registered components based on the intent data(**implicit intents).**

Explicit intents explicitly define the component which should be called by the Android system, by using the Java class as identifier. Explicit intents are typically used within an application as the classes in an application are controlled by the application developer.

The following shows how to create an explicit intent and send it to the Android system to start an activity.

Intent i = **new** Intent(this, ActivityTwo.class);

i.putExtra("Value1", "This value one for ActivityTwo ");

i.putExtra("Value2", "This value two ActivityTwo");

Implicit intents specify the action which should be performed and optionally data which provides content for the action. If an implicit intent is sent to the Android system, it searches for all components which are registered for the specific action and the fitting data type.

If only one component is found, Android starts this component directly.

If several components are identified by the Android system, the user will get a selection dialog and can decide which component should be used for the intent.

For example, the following tells the Android system to view a webpage. All installed web browsers should be registered to the corresponding intent data via an intent filter.

Intent i = **new** Intent(Intent.ACTION\_VIEW, Uri.parse("http://www.vogella.com"));

startActivity(i);

### [Determine valid intent receivers](http://www.vogella.com/tutorials/AndroidIntent/article.html#determine-valid-intent-receivers)

Sometimes you want to determine if a component has registered for an intent. For example, you want to check if a certain intent receiver is available and in case a component is available, you enable a functionality in your application.

This check can be done via the **PackageManager class**.

The following example code checks if a component has registered for a certain intent. Construct your intent as you are desired to trigger it and pass it to the following method.

**public** **static** **boolean** isIntentAvailable(Context ctx, Intent intent) {

**final** PackageManager mgr = ctx.getPackageManager();

List<ResolveInfo> list =

mgr.queryIntentActivities(intent,

PackageManager.MATCH\_DEFAULT\_ONLY);

**return** list.size() > 0;

}

Based on the result you can adjust your application. For example, you could disable or hide certain menu items.

### [Intents as event triggers](http://www.vogella.com/tutorials/AndroidIntent/article.html#intents-as-event-triggers)

Intents can be used to send broadcast messages into the Android system. A broadcast receiver can register to an event and is notified if such an event is sent.

Your application can register to system events, e.g., a new email has arrived, system boot is complete or a phone call is received and react accordingly.

## [Data transfer between activities](http://www.vogella.com/tutorials/AndroidIntent/article.html#data-transfer-between-activities)

### [Data transfer to the target component](http://www.vogella.com/tutorials/AndroidIntent/article.html#data-transfer-to-the-target-component)

An intent contains certain header data, e.g., the desired action, the type, etc. Optionally an intent can also contain additional data based on an instance of the Bundle class which can be retrieved from the intent via the getExtras() method.

You can also add data directly to the Bundle via the overloaded putExtra()methods of the Intent objects. Extras are key/value pairs. The key is always of type String. As value you can use the primitive data types (int, float, …​) plus objects of type String, Bundle, Parcelable and Serializable.

The receiving component can access this information via the getAction()and getData() methods on the Intent object.

This Intent object can be retrieved via the getIntent() method.

The component which receives the intent can use the getIntent().getExtras() method call to get the extra data. That is demonstrated in the following code snippet.

Bundle extras = getIntent().getExtras();

**if** (extras == null) {

**return**;

}

*// get data via the key*

String value1 = extras.getString(Intent.EXTRA\_TEXT);

**if** (value1 != null) {

*// do something with the data*

}

### [Example: Using the share intent](http://www.vogella.com/tutorials/AndroidIntent/article.html#example-using-the-share-intent)

Lots of Android applications allow you to share some data with other people, e.g., the Facebook, G+, Gmail and Twitter application. You can send data to one of these components. The following code snippet demonstrates the usage of such an intent within your application.

*// this runs, for example, after a button click*

Intent intent = **new** Intent(Intent.ACTION\_SEND);

intent.setType("text/plain");

intent.putExtra(android.content.Intent.EXTRA\_TEXT, "News for you!");

startActivity(intent);

### [Retrieving result data from a sub-activity](http://www.vogella.com/tutorials/AndroidIntent/article.html#retrieving-result-data-from-a-sub-activity)

An activity can be closed via the back button on the phone. In this case the finish() method is performed. If the activity was started with the startActivity(Intent) method call, the caller requires no result or feedback from the activity which now is closed.

If you start the activity with the startActivityForResult() method call, you expect feedback from the sub-activity. Once the sub-activity ends, the onActivityResult() method on the sub-activity is called and you can perform actions based on the result.

In the startActivityForResult() method call you can specify a result code to determine which activity you started. This result code is returned to you. The started activity can also set a result code which the caller can use to determine if the activity was cancelled or not.

An [Intent](https://developer.android.com/reference/android/content/Intent.html) is a messaging object you can use to request an action from another [app component](https://developer.android.com/guide/components/fundamentals.html#Components). Although intents facilitate communication between components in several ways, there are three fundamental use cases:

* **Starting an activity**

An [Activity](https://developer.android.com/reference/android/app/Activity.html) represents a single screen in an app. You can start a new instance of an [Activity](https://developer.android.com/reference/android/app/Activity.html) by passing an [Intent](https://developer.android.com/reference/android/content/Intent.html) to [startActivity()](https://developer.android.com/reference/android/content/Context.html#startActivity(android.content.Intent)). The [Intent](https://developer.android.com/reference/android/content/Intent.html) describes the activity to start and carries any necessary data.

If you want to receive a result from the activity when it finishes, call [startActivityForResult()](https://developer.android.com/reference/android/app/Activity.html#startActivityForResult(android.content.Intent, int)). Your activity receives the result as a separate [Intent](https://developer.android.com/reference/android/content/Intent.html) object in your activity's [onActivityResult()](https://developer.android.com/reference/android/app/Activity.html#onActivityResult(int, int, android.content.Intent)) callback. For more information, see the [Activities](https://developer.android.com/guide/components/activities.html) guide.

* **Starting a service**

A [Service](https://developer.android.com/reference/android/app/Service.html) is a component that performs operations in the background without a user interface. With Android 5.0 (API level 21) and later, you can start a service with [JobScheduler](https://developer.android.com/reference/android/app/job/JobScheduler.html). For more information about [JobScheduler](https://developer.android.com/reference/android/app/job/JobScheduler.html), see its [API-reference documentation](https://developer.android.com/reference/android/app/job/JobScheduler.html).

For versions earlier than Android 5.0 (API level 21), you can start a service by using methods of the[Service](https://developer.android.com/reference/android/app/Service.html) class. You can start a service to perform a one-time operation (such as downloading a file) by passing an [Intent](https://developer.android.com/reference/android/content/Intent.html) to [startService()](https://developer.android.com/reference/android/content/Context.html#startService(android.content.Intent)).

The [Intent](https://developer.android.com/reference/android/content/Intent.html) describes the service to start and carries any necessary data.

If the service is designed with a client-server interface, you can bind to the service from another component by passing an [Intent](https://developer.android.com/reference/android/content/Intent.html) to [bindService()](https://developer.android.com/reference/android/content/Context.html#bindService(android.content.Intent, android.content.ServiceConnection, int)). For more information, see the [Services](https://developer.android.com/guide/components/services.html) guide.

* **Delivering a broadcast**

A broadcast is a message that any app can receive. The system delivers various broadcasts for system events, such as when the system boots up or the device starts charging. You can deliver a broadcast to other apps by passing an [Intent](https://developer.android.com/reference/android/content/Intent.html) to [sendBroadcast()](https://developer.android.com/reference/android/content/Context.html#sendBroadcast(android.content.Intent)) or [sendOrderedBroadcast()](https://developer.android.com/reference/android/content/Context.html#sendOrderedBroadcast(android.content.Intent, java.lang.String)).

## Intent types

There are two types of intents:

* **Explicit intents** specify the component to start by name (the fully-qualified class name). You'll typically use an explicit intent to start a component in your own app, because you know the class name of the activity or service you want to start. For example, you can start a new activity in response to a user action or start a service to download a file in the background.
* **Implicit intents** do not name a specific component, but instead declare a general action to perform, which allows a component from another app to handle it. For example, if you want to show the user a location on a map, you can use an implicit intent to request that another capable app show a specified location on a map.

Topics to learn

* + - * 1. Intents

What is an Intent ...android.content.Intent class

Android Intent Messaging via Intent Objects - anatomy of an Intent object

Intent resolution - Implicit Intents and Explicit Intents

Using Intents with Activities - Intent Filters

Example - A Digital Clock Alternate Activity

Using Intents with Services

Using Intents with Broadcast Receivers

Pending Intents

Using Intent Object to invoke built-in app

What is Share Intent

Illustrated Example

* http://www.vogella.com/tutorials/AndroidIntent/article.html
* https://www.tutorialspoint.com/android/android\_intents\_filters.htm
* https://www.javatpoint.com/android-intent-tutorial
* https://developer.android.com/guide/components/intents-filters.html
* https://www.raywenderlich.com/103044/android-intents-tutorial
* http://programmerguru.com/android-tutorial/android-intent-example/
* http://o7planning.org/en/10425/android-intents-tutorial
* https://www.mkyong.com/android/android-activity-from-one-screen-to-another-screen/

Background

A [Service](https://developer.android.com/reference/android/app/Service.html) is an application component that can perform long-running operations in the background, and it does not provide a user interface. Another application component can start a service, and it continues to run in the background even if the user switches to another application. Additionally, a component can bind to a service to interact with it and even perform interprocess communication (IPC). For example, a service can handle network transactions, play music, perform file I/O, or interact with a content provider, all from the background.

These are the three different types of services:

**Scheduled**

A service is scheduled when an API such as the [JobScheduler](https://developer.android.com/reference/android/app/job/JobScheduler.html), introduced in Android 5.0 (API level 21), launches the service. You can use the [JobScheduler](https://developer.android.com/reference/android/app/job/JobScheduler.html) by registering jobs and specifying their requirements for network and timing. The system then gracefully schedules the jobs for execution at the appropriate times. The [JobScheduler](https://developer.android.com/reference/android/app/job/JobScheduler.html) provides many methods to define service-execution conditions.

**Note:** If your app targets Android 5.0 (API level 21), Google recommends that you use the [JobScheduler](https://developer.android.com/reference/android/app/job/JobScheduler.html) to execute background services. For more information about using this class, see the [JobScheduler](https://developer.android.com/reference/android/app/job/JobScheduler.html) reference documentation.

**Started**

A service is started when an application component (such as an activity) calls [startService()](https://developer.android.com/reference/android/content/Context.html#startService(android.content.Intent)). After it's started, a service can run in the background indefinitely, even if the component that started it is destroyed. Usually, a started service performs a single operation and does not return a result to the caller. For example, it can download or upload a file over the network. When the operation is complete, the service should stop itself.

**Bound**

A service is bound when an application component binds to it by calling [bindService()](https://developer.android.com/reference/android/content/Context.html#bindService(android.content.Intent, android.content.ServiceConnection, int)). A bound service offers a client-server interface that allows components to interact with the service, send requests, receive results, and even do so across processes with interprocess communication (IPC). A bound service runs only as long as another application component is bound to it. Multiple components can bind to the service at once, but when all of them unbind, the service is destroyed.

Although this documentation generally discusses started and bound services separately, your service can work both ways—it can be started (to run indefinitely) and also allow binding. It's simply a matter of whether you implement a couple of callback methods: [onStartCommand()](https://developer.android.com/reference/android/app/Service.html#onStartCommand(android.content.Intent, int, int)) to allow components to start it and [onBind()](https://developer.android.com/reference/android/app/Service.html#onBind(android.content.Intent)) to allow binding.

Regardless of whether your application is started, bound, or both, any application component can use the service (even from a separate application) in the same way that any component can use an activity—by starting it with an [Intent](https://developer.android.com/reference/android/content/Intent.html). However, you can declare the service as private in the manifest file and block access from other applications. This is discussed more in the section about [Declaring the service in the manifest](https://developer.android.com/guide/components/services.html#Declaring).

**Caution:** A service runs in the main thread of its hosting process; the service does **not** create its own thread and does **not** run in a separate process unless you specify otherwise. If your service is going to perform any CPU-intensive work or blocking operations, such as MP3 playback or networking, you should create a new thread within the service to complete that work. By using a separate thread, you can reduce the risk of Application Not Responding (ANR) errors, and the application's main thread can remain dedicated to user interaction with your activities.

## The basics

### Should you use a service or a thread?

A service is simply a component that can run in the background, even when the user is not interacting with your application, so you should create a service only if that is what you need.

If you must perform work outside of your main thread, but only while the user is interacting with your application, you should instead create a new thread. For example, if you want to play some music, but only while your activity is running, you might create a thread in [onCreate()](https://developer.android.com/reference/android/app/Activity.html#onCreate(android.os.Bundle)), start running it in [onStart()](https://developer.android.com/reference/android/app/Activity.html#onStart()), and stop it in [onStop()](https://developer.android.com/reference/android/app/Activity.html#onStop()). Also consider using [AsyncTask](https://developer.android.com/reference/android/os/AsyncTask.html) or [HandlerThread](https://developer.android.com/reference/android/os/HandlerThread.html) instead of the traditional [Thread](https://developer.android.com/reference/java/lang/Thread.html) class. See the [Processes and Threading](https://developer.android.com/guide/components/processes-and-threads.html#Threads) document for more information about threads.

Remember that if you do use a service, it still runs in your application's main thread by default, so you should still create a new thread within the service if it performs intensive or blocking operations.

To create a service, you must create a subclass of [Service](https://developer.android.com/reference/android/app/Service.html) or use one of its existing subclasses. In your implementation, you must override some callback methods that handle key aspects of the service lifecycle and provide a mechanism that allows the components to bind to the service, if appropriate. These are the most important callback methods that you should override:

[**onStartCommand()**](https://developer.android.com/reference/android/app/Service.html#onStartCommand(android.content.Intent, int, int))

The system invokes this method by calling [startService()](https://developer.android.com/reference/android/content/Context.html#startService(android.content.Intent)) when another component (such as an activity) requests that the service be started. When this method executes, the service is started and can run in the background indefinitely. If you implement this, it is your responsibility to stop the service when its work is complete by calling [stopSelf()](https://developer.android.com/reference/android/app/Service.html#stopSelf()) or [stopService()](https://developer.android.com/reference/android/content/Context.html#stopService(android.content.Intent)). If you only want to provide binding, you don't need to implement this method.

[**onBind()**](https://developer.android.com/reference/android/app/Service.html#onBind(android.content.Intent))

The system invokes this method by calling [bindService()](https://developer.android.com/reference/android/content/Context.html#bindService(android.content.Intent, android.content.ServiceConnection, int)) when another component wants to bind with the service (such as to perform RPC). In your implementation of this method, you must provide an interface that clients use to communicate with the service by returning an [IBinder](https://developer.android.com/reference/android/os/IBinder.html). You must always implement this method; however, if you don't want to allow binding, you should return null.

[**onCreate()**](https://developer.android.com/reference/android/app/Service.html#onCreate())

The system invokes this method to perform one-time setup procedures when the service is initially created (before it calls either [onStartCommand()](https://developer.android.com/reference/android/app/Service.html#onStartCommand(android.content.Intent, int, int)) or [onBind()](https://developer.android.com/reference/android/app/Service.html#onBind(android.content.Intent))). If the service is already running, this method is not called.

[**onDestroy()**](https://developer.android.com/reference/android/app/Service.html#onDestroy())

The system invokes this method when the service is no longer used and is being destroyed. Your service should implement this to clean up any resources such as threads, registered listeners, or receivers. This is the last call that the service receives.

If a component starts the service by calling [startService()](https://developer.android.com/reference/android/content/Context.html#startService(android.content.Intent)) (which results in a call to [onStartCommand()](https://developer.android.com/reference/android/app/Service.html#onStartCommand(android.content.Intent, int, int))), the service continues to run until it stops itself with [stopSelf()](https://developer.android.com/reference/android/app/Service.html#stopSelf()) or another component stops it by calling [stopService()](https://developer.android.com/reference/android/content/Context.html#stopService(android.content.Intent)).

If a component calls [bindService()](https://developer.android.com/reference/android/content/Context.html#bindService(android.content.Intent, android.content.ServiceConnection, int)) to create the service and [onStartCommand()](https://developer.android.com/reference/android/app/Service.html#onStartCommand(android.content.Intent, int, int)) is not called, the service runs only as long as the component is bound to it. After the service is unbound from all of its clients, the system destroys it.

The Android system force-stops a service only when memory is low and it must recover system resources for the activity that has user focus. If the service is bound to an activity that has user focus, it's less likely to be killed; if the service is declared to [run in the foreground](https://developer.android.com/guide/components/services.html#Foreground), it's rarely killed. If the service is started and is long-running, the system lowers its position in the list of background tasks over time, and the service becomes highly susceptible to killing—if your service is started, you must design it to gracefully handle restarts by the system. If the system kills your service, it restarts it as soon as resources become available, but this also depends on the value that you return from [onStartCommand()](https://developer.android.com/reference/android/app/Service.html#onStartCommand(android.content.Intent, int, int)). For more information about when the system might destroy a service, see the [Processes and Threading](https://developer.android.com/guide/components/processes-and-threads.html) document.

In the following sections, you'll see how you can create the [startService()](https://developer.android.com/reference/android/content/Context.html#startService(android.content.Intent)) and [bindService()](https://developer.android.com/reference/android/content/Context.html#bindService(android.content.Intent, android.content.ServiceConnection, int)) service methods, as well as how to use them from other application components.

### Declaring a service in the manifest

You must declare all services in your application's manifest file, just as you do for activities and other components.

To declare your service, add a [<service>](https://developer.android.com/guide/topics/manifest/service-element.html) element as a child of the [<application>](https://developer.android.com/guide/topics/manifest/application-element.html) element.

Here is an example:

<manifest ... >  
  ...  
  <application ... >  
      <service android:name=".ExampleService" />  
      ...  
  </application>  
</manifest>

See the [<service>](https://developer.android.com/guide/topics/manifest/service-element.html) element reference for more information about declaring your service in the manifest.

There are other attributes that you can include in the [<service>](https://developer.android.com/guide/topics/manifest/service-element.html) element to define properties such as the permissions that are required to start the service and the process in which the service should run. The [android:name](https://developer.android.com/guide/topics/manifest/service-element.html#nm) attribute is the only required attribute—it specifies the class name of the service. After you publish your application, leave this name unchanged to avoid the risk of breaking code due to dependence on explicit intents to start or bind the service (read the blog post, [Things That Cannot Change](http://android-developers.blogspot.com/2011/06/things-that-cannot-change.html)).

**Caution**: To ensure that your app is secure, always use an explicit intent when starting a [Service](https://developer.android.com/reference/android/app/Service.html) and do not declare intent filters for your services. Using an implicit intent to start a service is a security hazard because you cannot be certain of the service that will respond to the intent, and the user cannot see which service starts. Beginning with Android 5.0 (API level 21), the system throws an exception if you call [bindService()](https://developer.android.com/reference/android/content/Context.html#bindService(android.content.Intent, android.content.ServiceConnection, int)) with an implicit intent.

You can ensure that your service is available to only your app by including the [android:exported](https://developer.android.com/guide/topics/manifest/service-element.html#exported) attribute and setting it to false. This effectively stops other apps from starting your service, even when using an explicit intent.

**Note**: Users can see what services are running on their device. If they see a service that they don't recognize or trust, they can stop the service. In order to avoid having your service stopped accidentally by users, you need to add the [android:description](https://developer.android.com/guide/topics/manifest/service-element.html#desc) attribute to the [<service>](https://developer.android.com/guide/topics/manifest/service-element.html) element in your app manifest. In the description, provide a short sentence explaining what the service does and what benefits it provides.

## Creating a started service

A started service is one that another component starts by calling [startService()](https://developer.android.com/reference/android/content/Context.html#startService(android.content.Intent)), which results in a call to the service's [onStartCommand()](https://developer.android.com/reference/android/app/Service.html#onStartCommand(android.content.Intent, int, int)) method.

When a service is started, it has a lifecycle that's independent of the component that started it. The service can run in the background indefinitely, even if the component that started it is destroyed. As such, the service should stop itself when its job is complete by calling [stopSelf()](https://developer.android.com/reference/android/app/Service.html#stopSelf()), or another component can stop it by calling [stopService()](https://developer.android.com/reference/android/content/Context.html#stopService(android.content.Intent)).

An application component such as an activity can start the service by calling [startService()](https://developer.android.com/reference/android/content/Context.html#startService(android.content.Intent)) and passing an [Intent](https://developer.android.com/reference/android/content/Intent.html) that specifies the service and includes any data for the service to use. The service receives this [Intent](https://developer.android.com/reference/android/content/Intent.html) in the [onStartCommand()](https://developer.android.com/reference/android/app/Service.html#onStartCommand(android.content.Intent, int, int)) method.

For instance, suppose an activity needs to save some data to an online database. The activity can start a companion service and deliver it the data to save by passing an intent to [startService()](https://developer.android.com/reference/android/content/Context.html#startService(android.content.Intent)). The service receives the intent in [onStartCommand()](https://developer.android.com/reference/android/app/Service.html#onStartCommand(android.content.Intent, int, int)), connects to the Internet, and performs the database transaction. When the transaction is complete, the service stops itself and is destroyed.

**Caution:** A service runs in the same process as the application in which it is declared and in the main thread of that application by default. If your service performs intensive or blocking operations while the user interacts with an activity from the same application, the service slows down activity performance. To avoid impacting application performance, start a new thread inside the service.

Traditionally, there are two classes you can extend to create a started service:

[Service](https://developer.android.com/reference/android/app/Service.html)

This is the base class for all services. When you extend this class, it's important to create a new thread in which the service can complete all of its work; the service uses your application's main thread by default, which can slow the performance of any activity that your application is running.

[IntentService](https://developer.android.com/reference/android/app/IntentService.html)

This is a subclass of [Service](https://developer.android.com/reference/android/app/Service.html) that uses a worker thread to handle all of the start requests, one at a time. This is the best option if you don't require that your service handle multiple requests simultaneously. Implement [onHandleIntent()](https://developer.android.com/reference/android/app/IntentService.html#onHandleIntent(android.content.Intent)), which receives the intent for each start request so that you can complete the background work.

The following sections describe how you can implement your service using either one for these classes.

### Extending the IntentService class

Because most of the started services don't need to handle multiple requests simultaneously (which can actually be a dangerous multi-threading scenario), it's best that you implement your service using the [IntentService](https://developer.android.com/reference/android/app/IntentService.html) class.

The [IntentService](https://developer.android.com/reference/android/app/IntentService.html) class does the following:

* It creates a default worker thread that executes all of the intents that are delivered to [onStartCommand()](https://developer.android.com/reference/android/app/Service.html#onStartCommand(android.content.Intent, int, int)), separate from your application's main thread.
* Creates a work queue that passes one intent at a time to your [onHandleIntent()](https://developer.android.com/reference/android/app/IntentService.html#onHandleIntent(android.content.Intent)) implementation, so you never have to worry about multi-threading.
* Stops the service after all of the start requests are handled, so you never have to call [stopSelf()](https://developer.android.com/reference/android/app/Service.html#stopSelf()).
* Provides a default implementation of [onBind()](https://developer.android.com/reference/android/app/IntentService.html#onBind(android.content.Intent)) that returns null.
* Provides a default implementation of [onStartCommand()](https://developer.android.com/reference/android/app/IntentService.html#onStartCommand(android.content.Intent, int, int)) that sends the intent to the work queue and then to your [onHandleIntent()](https://developer.android.com/reference/android/app/IntentService.html#onHandleIntent(android.content.Intent)) implementation.

To complete the work that is provided by the client, implement [onHandleIntent()](https://developer.android.com/reference/android/app/IntentService.html#onHandleIntent(android.content.Intent)). However, you also need to provide a small constructor for the service.

Here's an example implementation of [IntentService](https://developer.android.com/reference/android/app/IntentService.html):

public class HelloIntentService extends IntentService {  
  
  /\*\*  
   \* A constructor is required, and must call the super [IntentService(String)](https://developer.android.com/reference/android/app/IntentService.html#IntentService(java.lang.String))  
   \* constructor with a name for the worker thread.  
   \*/  
  public HelloIntentService() {  
      super("HelloIntentService");  
  }  
  
  /\*\*  
   \* The IntentService calls this method from the default worker thread with  
   \* the intent that started the service. When this method returns, IntentService  
   \* stops the service, as appropriate.  
   \*/  
  @Override  
  protected void onHandleIntent(Intent intent) {  
      // Normally we would do some work here, like download a file.  
      // For our sample, we just sleep for 5 seconds.  
      try {  
          Thread.sleep(5000);  
      } catch (InterruptedException e) {  
          // Restore interrupt status.  
          Thread.currentThread().interrupt();  
      }  
  }  
}

That's all you need: a constructor and an implementation of [onHandleIntent()](https://developer.android.com/reference/android/app/IntentService.html#onHandleIntent(android.content.Intent)).

If you decide to also override other callback methods, such as [onCreate()](https://developer.android.com/reference/android/app/IntentService.html#onCreate()), [onStartCommand()](https://developer.android.com/reference/android/app/IntentService.html#onStartCommand(android.content.Intent, int, int)), or [onDestroy()](https://developer.android.com/reference/android/app/IntentService.html#onDestroy()), be sure to call the super implementation so that the [IntentService](https://developer.android.com/reference/android/app/IntentService.html) can properly handle the life of the worker thread.

For example, [onStartCommand()](https://developer.android.com/reference/android/app/IntentService.html#onStartCommand(android.content.Intent, int, int)) must return the default implementation, which is how the intent is delivered to [onHandleIntent()](https://developer.android.com/reference/android/app/IntentService.html#onHandleIntent(android.content.Intent)):

@Override  
public int onStartCommand(Intent intent, int flags, int startId) {  
    Toast.makeText(this, "service starting", Toast.LENGTH\_SHORT).show();  
    return super.onStartCommand(intent,flags,startId);  
}

Besides [onHandleIntent()](https://developer.android.com/reference/android/app/IntentService.html#onHandleIntent(android.content.Intent)), the only method from which you don't need to call the super class is [onBind()](https://developer.android.com/reference/android/app/IntentService.html#onBind(android.content.Intent)). You need to implement this only if your service allows binding.

In the next section, you'll see how the same kind of service is implemented when extending the base [Service](https://developer.android.com/reference/android/app/Service.html) class, which uses more code, but might be appropriate if you need to handle simultaneous start requests.

### Extending the Service class

Using [IntentService](https://developer.android.com/reference/android/app/IntentService.html) makes your implementation of a started service very simple. If, however, you require your service to perform multi-threading (instead of processing start requests through a work queue), you can extend the [Service](https://developer.android.com/reference/android/app/Service.html) class to handle each intent.

For comparison, the following example code shows an implementation of the [Service](https://developer.android.com/reference/android/app/Service.html) class that performs the same work as the previous example using [IntentService](https://developer.android.com/reference/android/app/IntentService.html). That is, for each start request, it uses a worker thread to perform the job and processes only one request at a time.

public class HelloService extends Service {  
  private Looper mServiceLooper;  
  private ServiceHandler mServiceHandler;  
  
  // Handler that receives messages from the thread  
  private final class ServiceHandler extends Handler {  
      public ServiceHandler(Looper looper) {  
          super(looper);  
      }  
      @Override  
      public void handleMessage(Message msg) {  
          // Normally we would do some work here, like download a file.  
          // For our sample, we just sleep for 5 seconds.  
          try {  
              Thread.sleep(5000);  
          } catch (InterruptedException e) {  
              // Restore interrupt status.  
              Thread.currentThread().interrupt();  
          }  
          // Stop the service using the startId, so that we don't stop  
          // the service in the middle of handling another job  
          stopSelf(msg.arg1);  
      }  
  }  
  
  @Override  
  public void onCreate() {  
    // Start up the thread running the service.  Note that we create a  
    // separate thread because the service normally runs in the process's  
    // main thread, which we don't want to block.  We also make it  
    // background priority so CPU-intensive work will not disrupt our UI.  
    HandlerThread thread = new HandlerThread("ServiceStartArguments",  
            Process.THREAD\_PRIORITY\_BACKGROUND);  
    thread.start();  
  
    // Get the HandlerThread's Looper and use it for our Handler  
    mServiceLooper = thread.getLooper();  
    mServiceHandler = new ServiceHandler(mServiceLooper);  
  }  
  
  @Override  
  public int onStartCommand(Intent intent, int flags, int startId) {  
      Toast.makeText(this, "service starting", Toast.LENGTH\_SHORT).show();  
  
      // For each start request, send a message to start a job and deliver the  
      // start ID so we know which request we're stopping when we finish the job  
      Message msg = mServiceHandler.obtainMessage();  
      msg.arg1 = startId;  
      mServiceHandler.sendMessage(msg);  
  
      // If we get killed, after returning from here, restart  
      return START\_STICKY;  
  }  
  
  @Override  
  public IBinder onBind(Intent intent) {  
      // We don't provide binding, so return null  
      return null;  
  }  
  
  @Override  
  public void onDestroy() {  
    Toast.makeText(this, "service done", Toast.LENGTH\_SHORT).show();  
  }  
}

As you can see, it's a lot more work than using [IntentService](https://developer.android.com/reference/android/app/IntentService.html).

However, because you handle each call to [onStartCommand()](https://developer.android.com/reference/android/app/Service.html#onStartCommand(android.content.Intent, int, int)) yourself, you can perform multiple requests simultaneously. That's not what this example does, but if that's what you want, you can create a new thread for each request and run them right away instead of waiting for the previous request to finish.

Notice that the [onStartCommand()](https://developer.android.com/reference/android/app/Service.html#onStartCommand(android.content.Intent, int, int)) method must return an integer. The integer is a value that describes how the system should continue the service in the event that the system kills it. The default implementation for [IntentService](https://developer.android.com/reference/android/app/IntentService.html) handles this for you, but you are able to modify it. The return value from [onStartCommand()](https://developer.android.com/reference/android/app/Service.html#onStartCommand(android.content.Intent, int, int)) must be one of the following constants:

[**START\_NOT\_STICKY**](https://developer.android.com/reference/android/app/Service.html#START_NOT_STICKY)

If the system kills the service after [onStartCommand()](https://developer.android.com/reference/android/app/Service.html#onStartCommand(android.content.Intent, int, int)) returns, do not recreate the service unless there are pending intents to deliver. This is the safest option to avoid running your service when not necessary and when your application can simply restart any unfinished jobs.

[**START\_STICKY**](https://developer.android.com/reference/android/app/Service.html#START_STICKY)

If the system kills the service after [onStartCommand()](https://developer.android.com/reference/android/app/Service.html#onStartCommand(android.content.Intent, int, int)) returns, recreate the service and call [onStartCommand()](https://developer.android.com/reference/android/app/Service.html#onStartCommand(android.content.Intent, int, int)), but do not redeliver the last intent. Instead, the system calls [onStartCommand()](https://developer.android.com/reference/android/app/Service.html#onStartCommand(android.content.Intent, int, int)) with a null intent unless there are pending intents to start the service. In that case, those intents are delivered. This is suitable for media players (or similar services) that are not executing commands but are running indefinitely and waiting for a job.

[**START\_REDELIVER\_INTENT**](https://developer.android.com/reference/android/app/Service.html#START_REDELIVER_INTENT)

If the system kills the service after [onStartCommand()](https://developer.android.com/reference/android/app/Service.html#onStartCommand(android.content.Intent, int, int)) returns, recreate the service and call [onStartCommand()](https://developer.android.com/reference/android/app/Service.html#onStartCommand(android.content.Intent, int, int)) with the last intent that was delivered to the service. Any pending intents are delivered in turn. This is suitable for services that are actively performing a job that should be immediately resumed, such as downloading a file.

For more details about these return values, see the linked reference documentation for each constant.

### Starting a service

You can start a service from an activity or other application component by passing an [Intent](https://developer.android.com/reference/android/content/Intent.html) (specifying the service to start) to [startService()](https://developer.android.com/reference/android/content/Context.html#startService(android.content.Intent)). The Android system calls the service's [onStartCommand()](https://developer.android.com/reference/android/app/Service.html#onStartCommand(android.content.Intent, int, int)) method and passes it the [Intent](https://developer.android.com/reference/android/content/Intent.html). **Note**: Never call [onStartCommand()](https://developer.android.com/reference/android/app/Service.html#onStartCommand(android.content.Intent, int, int)) directly.

For example, an activity can start the example service in the previous section (HelloService) using an explicit intent with [startService()](https://developer.android.com/reference/android/content/Context.html#startService(android.content.Intent)), as shown here:

Intent intent = new Intent(this, HelloService.class);  
startService(intent);

The [startService()](https://developer.android.com/reference/android/content/Context.html#startService(android.content.Intent)) method returns immediately, and the Android system calls the service's [onStartCommand()](https://developer.android.com/reference/android/app/Service.html#onStartCommand(android.content.Intent, int, int)) method. If the service is not already running, the system first calls [onCreate()](https://developer.android.com/reference/android/app/Service.html#onCreate()), and then it calls [onStartCommand()](https://developer.android.com/reference/android/app/Service.html#onStartCommand(android.content.Intent, int, int)).

If the service does not also provide binding, the intent that is delivered with [startService()](https://developer.android.com/reference/android/content/Context.html#startService(android.content.Intent)) is the only mode of communication between the application component and the service. However, if you want the service to send a result back, the client that starts the service can create a [PendingIntent](https://developer.android.com/reference/android/app/PendingIntent.html) for a broadcast (with [getBroadcast()](https://developer.android.com/reference/android/app/PendingIntent.html#getBroadcast(android.content.Context, int, android.content.Intent, int))) and deliver it to the service in the [Intent](https://developer.android.com/reference/android/content/Intent.html) that starts the service. The service can then use the broadcast to deliver a result.

Multiple requests to start the service result in multiple corresponding calls to the service's [onStartCommand()](https://developer.android.com/reference/android/app/Service.html#onStartCommand(android.content.Intent, int, int)). However, only one request to stop the service (with [stopSelf()](https://developer.android.com/reference/android/app/Service.html#stopSelf()) or [stopService()](https://developer.android.com/reference/android/content/Context.html#stopService(android.content.Intent))) is required to stop it.

### Stopping a service

A started service must manage its own lifecycle. That is, the system does not stop or destroy the service unless it must recover system memory and the service continues to run after [onStartCommand()](https://developer.android.com/reference/android/app/Service.html#onStartCommand(android.content.Intent, int, int)) returns. The service must stop itself by calling [stopSelf()](https://developer.android.com/reference/android/app/Service.html#stopSelf()), or another component can stop it by calling [stopService()](https://developer.android.com/reference/android/content/Context.html#stopService(android.content.Intent)).

Once requested to stop with [stopSelf()](https://developer.android.com/reference/android/app/Service.html#stopSelf()) or [stopService()](https://developer.android.com/reference/android/content/Context.html#stopService(android.content.Intent)), the system destroys the service as soon as possible.

If your service handles multiple requests to [onStartCommand()](https://developer.android.com/reference/android/app/Service.html#onStartCommand(android.content.Intent, int, int)) concurrently, you shouldn't stop the service when you're done processing a start request, as you might have received a new start request (stopping at the end of the first request would terminate the second one). To avoid this problem, you can use [stopSelf(int)](https://developer.android.com/reference/android/app/Service.html#stopSelf(int)) to ensure that your request to stop the service is always based on the most recent start request. That is, when you call [stopSelf(int)](https://developer.android.com/reference/android/app/Service.html#stopSelf(int)), you pass the ID of the start request (the startId delivered to [onStartCommand()](https://developer.android.com/reference/android/app/Service.html#onStartCommand(android.content.Intent, int, int))) to which your stop request corresponds. Then, if the service receives a new start request before you are able to call [stopSelf(int)](https://developer.android.com/reference/android/app/Service.html#stopSelf(int)), the ID does not match and the service does not stop.

**Caution:** To avoid wasting system resources and consuming battery power, ensure that your application stops its services when it's done working. If necessary, other components can stop the service by calling [stopService()](https://developer.android.com/reference/android/content/Context.html#stopService(android.content.Intent)). Even if you enable binding for the service, you must always stop the service yourself if it ever receives a call to [onStartCommand()](https://developer.android.com/reference/android/app/Service.html#onStartCommand(android.content.Intent, int, int)).

For more information about the lifecycle of a service, see the section below about [Managing the Lifecycle of a Service](https://developer.android.com/guide/components/services.html#Lifecycle).

## Creating a bound service

A bound service is one that allows application components to bind to it by calling [bindService()](https://developer.android.com/reference/android/content/Context.html#bindService(android.content.Intent, android.content.ServiceConnection, int)) to create a long-standing connection. It generally **doesn't allow** components to start it by calling [startService()](https://developer.android.com/reference/android/content/Context.html#startService(android.content.Intent)).

Create a bound service when you want to interact with the service from activities and other components in your application or to expose some of your application's functionality to other applications through interprocess communication (IPC).

To create a bound service, implement the [onBind()](https://developer.android.com/reference/android/app/Service.html#onBind(android.content.Intent)) callback method to return an [IBinder](https://developer.android.com/reference/android/os/IBinder.html) that defines the interface for communication with the service. Other application components can then call [bindService()](https://developer.android.com/reference/android/content/Context.html#bindService(android.content.Intent, android.content.ServiceConnection, int)) to retrieve the interface and begin calling methods on the service. The service lives only to serve the application component that is bound to it, so when there are no components bound to the service, the system destroys it. You do not need to stop a bound service in the same way that you must when the service is started through [onStartCommand()](https://developer.android.com/reference/android/app/Service.html#onStartCommand(android.content.Intent, int, int)).

To create a bound service, you must define the interface that specifies how a client can communicate with the service. This interface between the service and a client must be an implementation of [IBinder](https://developer.android.com/reference/android/os/IBinder.html) and is what your service must return from the [onBind()](https://developer.android.com/reference/android/app/Service.html#onBind(android.content.Intent)) callback method. After the client receives the [IBinder](https://developer.android.com/reference/android/os/IBinder.html), it can begin interacting with the service through that interface.

Multiple clients can bind to the service simultaneously. When a client is done interacting with the service, it calls [unbindService()](https://developer.android.com/reference/android/content/Context.html#unbindService(android.content.ServiceConnection)) to unbind. When there are no clients bound to the service, the system destroys the service.

There are multiple ways to implement a bound service, and the implementation is more complicated than a started service. For these reasons, the bound service discussion appears in a separate document about [Bound Services](https://developer.android.com/guide/components/bound-services.html).

## Sending notifications to the user

When a service is running, it can notify the user of events using [Toast Notifications](https://developer.android.com/guide/topics/ui/notifiers/toasts.html) or [Status Bar Notifications](https://developer.android.com/guide/topics/ui/notifiers/notifications.html).

A **toast notification** is a message that appears on the surface of the current window for only a moment before disappearing. A status bar notification provides an icon in the status bar with a message, which the user can select in order to take an action (such as start an activity).

Usually, a **status bar notification** is the best technique to use when background work such as a file download has completed, and the user can now act on it. When the user selects the notification from the expanded view, the notification can start an activity (such as to display the downloaded file).

See the [Toast Notifications](https://developer.android.com/guide/topics/ui/notifiers/toasts.html) or [Status Bar Notifications](https://developer.android.com/guide/topics/ui/notifiers/notifications.html) developer guides for more information.

## Running a service in the foreground

A foreground service is a service that the user is actively aware of and is not a candidate for the system to kill when low on memory. A foreground service must provide a notification for the status bar, which is placed under the Ongoing heading. This means that the notification cannot be dismissed unless the service is either stopped or removed from the foreground.

For example, a music player that plays music from a service should be set to run in the foreground, because the user is explicitly aware of its operation. The notification in the status bar might indicate the current song and allow the user to launch an activity to interact with the music player.

To request that your service run in the foreground, call [startForeground()](https://developer.android.com/reference/android/app/Service.html#startForeground(int, android.app.Notification)). This method takes two parameters: an integer that uniquely identifies the notification and the [Notification](https://developer.android.com/reference/android/app/Notification.html) for the status bar. **Caution:** The integer ID that you give to [startForeground()](https://developer.android.com/reference/android/app/Service.html#startForeground(int, android.app.Notification)) must not be 0.

To remove the service from the foreground, call [stopForeground()](https://developer.android.com/reference/android/app/Service.html#stopForeground(boolean)). This method takes a boolean, which indicates whether to remove the status bar notification as well. This method does not stop the service. However, if you stop the service while it's still running in the foreground, the notification is also removed.

For more information about notifications, see [Creating Status Bar Notifications](https://developer.android.com/guide/topics/ui/notifiers/notifications.html).

## Managing the lifecycle of a service

The lifecycle of a service is much simpler than that of an activity. However, it's even more important that you pay close attention to how your service is created and destroyed because a service can run in the background without the user being aware.

The service lifecycle—from when it's created to when it's destroyed—can follow either of these two paths:

* A started service

The service is created when another component calls [startService()](https://developer.android.com/reference/android/content/Context.html#startService(android.content.Intent)). The service then runs indefinitely and must stop itself by calling [stopSelf()](https://developer.android.com/reference/android/app/Service.html#stopSelf()). Another component can also stop the service by calling [stopService()](https://developer.android.com/reference/android/content/Context.html#stopService(android.content.Intent)). When the service is stopped, the system destroys it.

* A bound service

The service is created when another component (a client) calls [bindService()](https://developer.android.com/reference/android/content/Context.html#bindService(android.content.Intent, android.content.ServiceConnection, int)). The client then communicates with the service through an [IBinder](https://developer.android.com/reference/android/os/IBinder.html)interface. The client can close the connection by calling [unbindService()](https://developer.android.com/reference/android/content/Context.html#unbindService(android.content.ServiceConnection)). Multiple clients can bind to the same service and when all of them unbind, the system destroys the service. The service does not need to stop itself.

These two paths are not entirely separate. You can bind to a service that is already started with [startService()](https://developer.android.com/reference/android/content/Context.html#startService(android.content.Intent)). For example, you can start a background music service by calling [startService()](https://developer.android.com/reference/android/content/Context.html#startService(android.content.Intent)) with an [Intent](https://developer.android.com/reference/android/content/Intent.html) that identifies the music to play. Later, possibly when the user wants to exercise some control over the player or get information about the current song, an activity can bind to the service by calling [bindService()](https://developer.android.com/reference/android/content/Context.html#bindService(android.content.Intent, android.content.ServiceConnection, int)). In cases such as this, [stopService()](https://developer.android.com/reference/android/content/Context.html#stopService(android.content.Intent)) or [stopSelf()](https://developer.android.com/reference/android/app/Service.html#stopSelf()) doesn't actually stop the service until all of the clients unbind.

### Implementing the lifecycle callbacks

Like an activity, a service has lifecycle callback methods that you can implement to monitor changes in the service's state and perform work at the appropriate times.

Unlike the activity lifecycle callback methods, you are not required to call the superclass implementation of these callback methods.



Topics to learn

* + - * 1. Services ...Services and Background Tasks

Using Services in an App,

When and How to use a Service

Exploring Service Types

Lifecycle of a Service

Running in the background

Communicating with Services

Creating a Self-Contained Service

Using a Foreground Service

Using an IntentService

Illustrated Example

* https://developer.android.com/guide/components/services.html
* https://www.tutorialspoint.com/android/android\_services.htm
* https://www.javatpoint.com/android-service-tutorial
* http://www.survivingwithandroid.com/2014/01/android-service-tutorial-2.html
* http://www.techotopia.com/index.php/Implementing\_an\_Android\_Started\_Service\_%E2%80%93\_A\_Worked\_Example
* http://programmerguru.com/android-tutorial/android-intent-example/
* https://www.mkyong.com/android/android-activity-from-one-screen-to-another-screen/

Background

**Broadcast Receivers**simply respond to broadcast messages from other applications or from the system itself. These messages are sometime called events or intents.

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.

There are following two important steps to make BroadcastReceiver works for the system broadcasted intents −

* Creating the Broadcast Receiver.
* Registering Broadcast Receiver

There is **one additional steps in case you are going to implement your custom intents** then you will have to create and broadcast those intents.

Android apps can **send or receive** broadcast messages from the Android system and other Android apps, similar to the [publish-subscribe](https://en.wikipedia.org/wiki/Publish%E2%80%93subscribe_pattern) design pattern.

These broadcasts are sent when an event of interest occurs.

For example, the Android system sends broadcasts when various system events occur, such as when the system boots up or the device starts charging.

Apps can also send custom broadcasts, for example, to notify other apps of something that they might be interested in (for example, some new data has been downloaded).

Apps can register to receive specific broadcasts. When a broadcast is sent, the system automatically routes broadcasts to apps that have subscribed to receive that particular type of broadcast.

A [**BroadcastReceiver**](http://developer.android.com/reference/android/content/BroadcastReceiver.html)is an Android app component that responds to system-wide broadcast announcements. Imagine an event like external power being connected/disconnected from the device, screen turning on/off, battery getting low or picture captured.

**All these events originate from the system.** In fact apps themselves can also initiate broadcasts – for example the SMS app broadcasting that an SMS has being received and let other apps know about this event so that they can trigger some action.

**Unlike Activities broadcast receivers** do not have any user interface but may create a status bar notification. It is intended to do minimal amount of work and can delegate hardcore jobs to Services. It receives an Intent object, so if you’ve read my previous articles on [Intents](http://codetheory.in/android-intents/) and [Intent Filters](http://codetheory.in/android-intent-filters/) you’ll have an easy time learning it.

So **the gist is broadcast receivers** are like dormant app components that can register for various system or application events (intents). Once any of those events occur the system notifies all the registered broadcast receivers and brings them up into action which could be notifying the user or perform some other job.

The registration is done in the manifest file using intent filters (static) but can also be done programatically (dynamic).

Topics to learn

* + - * 1. Broadcast Receiver

What is a Broadcast Receiver and the Receiver() method

Broadcasting events with Intents

Listening to Broadcasts with Broadcast Receivers

Broadcasting Sticky and Ordered Intents

Native Android Broadcast actions

Local Broadcast Receivers

Normal and Ordered Broadcasts

Directed Broadcasts

Enabling and disabling receivers

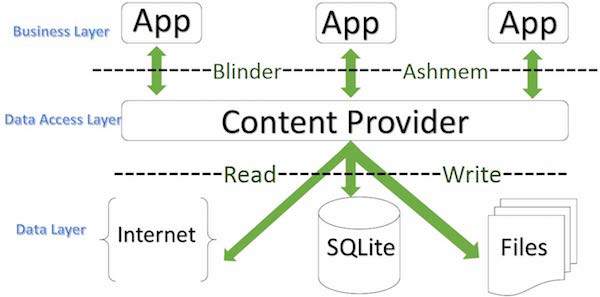
System Broadcast Intents

Illustrated Example

* https://www.tutorialspoint.com/android/android\_broadcast\_receivers.htm
* http://www.vogella.com/tutorials/AndroidBroadcastReceiver/article.html
* https://developer.android.com/reference/android/content/BroadcastReceiver.html
* https://developer.android.com/guide/components/broadcasts.html
* http://www.journaldev.com/10356/android-broadcastreceiver-example-tutorial
* http://codetheory.in/android-broadcast-receivers/
* https://examples.javacodegeeks.com/android/core/content/broadcastreceiver/android-broadcast-receivers-example/

Background

A content provider component supplies data from one application to others on request. Such requests are handled by the methods of the ContentResolver class. A content provider can use different ways to store its data and the data can be stored in a database, in files, or even over a network.



Content providers let you centralize content in one place and have many different applications access it as needed. A content provider behaves very much like a database where you can query it, edit its content, as well as add or delete content using insert(), update(), delete(), and query() methods. In most cases this data is stored in an **SQlite** database.

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.

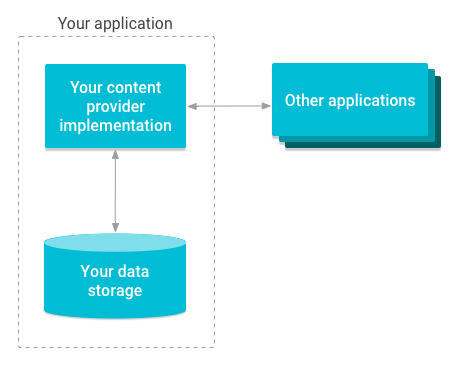
public class My Application extends ContentProvider {

}

A content provider manages access to a central repository of data. You implement a provider as one or more classes in an Android application, along with elements in the manifest file. One of your classes implements a subclass [ContentProvider](https://developer.android.com/reference/android/content/ContentProvider.html), which is the interface between your provider and other applications. Although content providers are meant to make data available to other applications, you may of course have activities in your application that allow the user to query and modify the data managed by your provider.

The rest of this topic is a basic list of steps for building a content provider and a list of APIs to use.

A content provider manages access to a central repository of data. A provider is part of an Android application, which often provides its own UI for working with the data. However, content providers are primarily intended to be used by other applications, which access the provider using a provider client object. Together, providers and provider clients offer a consistent, standard interface to data that also handles inter-process communication and secure data access.



Implementing a content provider involves always the following steps:

1. Create a class that extends ContentProvider
2. Create a contract class
3. Create the UriMatcher definition
4. Implement the onCreate() method
5. Implement the getType() method
6. Implement the CRUD methods
7. Add the content provider to your AndroidManifest.xml

Based on a sample project I will expand on all of these topics. The sample project is a simple app that you can use to manage lent items.

For the sake of this sample I use a very simple data model, which consists of only two tables: items and photos.

While the table structure would allow to add multiple photos to one item, the app doesn’t support it. In my arbitrary use case there exists at most one photo per lent item.

### Create a class that extends ContentProvider

You start by sub-classing [ContentProvider](http://developer.android.com/reference/android/content/ContentProvider.html). Since ContentProvider is an abstract class you have to implement the six abstract methods. These methods are explained in detail later on, for now simply use the stubs created by the IDE of your choice.

|  |  |
| --- | --- |
| Method | Usage |
| onCreate() | Prepares the content provider |
| getType(Uri) | Returns the MIME type for this URI |
| delete(Uri uri, String selection, String[] selectionArgs) | Deletes records |
| insert(Uri uri, ContentValues values) | Adds records |
| query(Uri uri, String[] projection, String selection, String[] selectionArgs, String sortOrder) | Return records based on selection criteria |
| update(Uri uri, ContentValues values, String selection, String[] selectionArgs) | Modifies data |
| **The abstract methods you have to implement** | |

### Create a contract class

Up to now your code is still missing most of the functionality. But before implementing the CRUD methods you should think about your role as a provider. Content providers by its very definition provide data to clients.

* Those clients need to know how to access your data.
* And you should treat your URIs and authority like an API.
* You basically enter into a contract with your client.
* And your public API should reflect this.

Thus the official Android documentation recommends to [create a contract class](http://developer.android.com/guide/topics/providers/content-provider-creating.html#ContractClass). This class defines all publicly available elements, like the authority, the content URIs of your tables, the columns, the content types and also any intents your app offers in addition to your provider.

This class is your public API. What you define here is what clients can use. It’s also the abstraction you provide. You can do behind the scenes whatever you like. The client won’t notice. You can change the data structure without problems – if your contract class remains unchanged.

The downside is: You shouldn’t change the contract in any way that might break existing clients.

It’s a contract after all

If you really think you must get rid of something you provided earlier on, use analytics to find out when a deprecated feature isn’t used anymore.

**SQLite is a open source SQL database t**hat stores data to a text file on a device. Android comes in with built in SQLite database implementation.

SQLite supports all the relational database features. In order to access this database, you don't need to establish any kind of connections for it like JDBC,ODBC etc

## Database - Package

## The main package is android.database.sqlite that contains the classes to manage your own databases

## Database - Creation

In order to create a database you just need to call this method openOrCreateDatabase with your database name and mode as a parameter. It returns an instance of SQLite database which you have to receive in your own object.

Its syntax is given below

SQLiteDatabase mydatabase = openOrCreateDatabase("your database name",MODE\_PRIVATE,null);

Apart from this , there are other functions available in the database package , that does this job.

**Saving data to a database** is ideal for repeating or structured data, such as contact information. This class assumes that you are familiar with SQL databases in general and helps you get started with SQLite databases on Android. The APIs you'll need to use a database on Android are available in the [android.database.sqlite](https://developer.android.com/reference/android/database/sqlite/package-summary.html) package.

One of the main principles of SQL databases is **the schema**: a formal declaration of how the database is organized. The schema is reflected in the SQL statements that you use to create your **database**. You may find it helpful to create a companion class, known **as a contract class**, which explicitly specifies the layout of your schema in a systematic and self-documenting way.

**A contract class** is a container for constants that define names for URIs, tables, and columns. The contract class allows you to use the same constants across all the other classes in the same package. This lets you change a column name in one place and have it propagate throughout your code.

A good way to organize a contract class is to put definitions that are **global** to your whole database in the root level of the class. Then create an inner class for each table that enumerates its columns.

---------------------------------------------------------------------------------------------------------------------------------------

Android provides many ways of storing data of an application. One of this way is called Shared Preferences. Shared Preferences allow you to save and retrieve data in the form of **key, value pair**.

**In order to use shared preferences, you have to call a method getSharedPreferences**() that returns a SharedPreference instance pointing to the file that contains the values of preferences.

SharedPreferences sharedpreferences = getSharedPreferences(MyPREFERENCES, Context.MODE\_PRIVATE);

The first parameter is the key and the **second parameter is the MODE**.

### Android Media Store

### [Playing sound](http://www.vogella.com/tutorials/AndroidMedia/article.html#playing-sound)

Android provides two main API’s for playing sounds. The first one via the SoundPool class and the other one via the MediaPlayer class. **SoundPoo**l can be used for small audio clips. It can repeat sounds and play several sounds simultaneously. The sound files played with SoundPool should not exceed 1 MB. SoundPool does load the file asynchronously. As of Android API8 it is possible to check if the loading is complete via a OnLoadCompleteListener.

Android supports different audio streams for different purposes. The phone volume button can be configured to control a specific audio stream, e.g. during a call the volume button allow increase / decrease the caller volume. To set the button to control the sound media stream set the audio type in your application.

context.setVolumeControlStream(AudioManager.STREAM\_MUSIC);

**MediaPlayer** is better suited for longer music and movies.

### [MediaRecorder](http://www.vogella.com/tutorials/AndroidMedia/article.html#mediarecorder)

The android.media.MediaRecorder class can be used to record audio and video. To use MediaRecorder you need to set the source device and the format.

### [Adding to the Media library](http://www.vogella.com/tutorials/AndroidMedia/article.html#adding-to-the-media-library)

You can add new Media to the Android media library. Via an Intent you can tell the media application on the device that new content is available. The following demonstrates that.

*// add new file to your media library*

ContentValues values = **new** ContentValues(4);

**long** current = System.currentTimeMillis();

values.put(MediaStore.Audio.Media.TITLE, "audio" + audiofile.getName());

values.put(MediaStore.Audio.Media.DATE\_ADDED, (**int**) (current / 1000));

values.put(MediaStore.Audio.Media.MIME\_TYPE, "audio/3gpp");

values.put(MediaStore.Audio.Media.DATA, audiofile.getAbsolutePath());

ContentResolver contentResolver = getContentResolver();

Uri base = MediaStore.Audio.Media.EXTERNAL\_CONTENT\_URI;

Uri newUri = contentResolver.insert(base, values);

*// Notifiy the media application on the device*

sendBroadcast(**new** Intent(Intent.ACTION\_MEDIA\_SCANNER\_SCAN\_FILE, newUri));

### [Supported formats](http://www.vogella.com/tutorials/AndroidMedia/article.html#supported-formats)

Android supports a variaty of formats. See<http://developer.android.com/guide/appendix/media-formats.html> - Supported Media Formats in the Android Developer Guide.

# How to [List of albums from MediaStore to adapter/listview](http://stackoverflow.com/questions/18926633/list-of-albums-from-mediastore-to-adapter-listview)

This gets you all albums:

public Cursor getAlbumAlbumcursor(Context context, Cursor cursor)

{

String where = null;

ContentResolver cr = context.getContentResolver();

final Uri uri = MediaStore.Audio.Albums.EXTERNAL\_CONTENT\_URI;

final String \_id = MediaStore.Audio.Albums.\_ID;

final String album\_id = MediaStore.Audio.Albums.ALBUM\_ID;

final String album\_name =MediaStore.Audio.Albums.ALBUM;

final String artist = MediaStore.Audio.Albums.ARTIST;

final String[]columns={\_id,album\_name, artist};

cursor = cr.query(uri,columns,where,null, null);

return cursor;

}

This gets you all tracks:

public Cursor getTrackTrackcursor(Context context, Cursor cursor)

{

final String track\_id = MediaStore.Audio.Media.\_ID;

final String track\_no =MediaStore.Audio.Media.TRACK;

final String track\_name =MediaStore.Audio.Media.TITLE;

final String artist = MediaStore.Audio.Media.ARTIST;

final String duration = MediaStore.Audio.Media.DURATION;

final String album = MediaStore.Audio.Media.ALBUM;

final String composer = MediaStore.Audio.Media.COMPOSER;

final String year = MediaStore.Audio.Media.YEAR;

final String path = MediaStore.Audio.Media.DATA;

Uri uri = MediaStore.Audio.Media.EXTERNAL\_CONTENT\_URI;

ContentResolver cr = context.getContentResolver();

final String[]columns={track\_id, track\_no, artist, track\_name,album, duration, path, year, composer};

cursor = cr.query(uri,columns,null,null,null);

return cursor;

}

This gets you all playlists:

public Cursor getandroidPlaylistcursor(Context context,Cursor cursor)

{

ContentResolver resolver = context.getContentResolver();

final Uri uri=MediaStore.Audio.Playlists.EXTERNAL\_CONTENT\_URI;

final String id = MediaStore.Audio.Playlists.\_ID;

final String name = MediaStore.Audio.Playlists.NAME;

final String[]columns = {id,name};

final String criteria = MediaStore.Audio.Playlists.NAME.length() + " > 0 " ;

final Cursor crplaylists = resolver.query(uri, columns, criteria, null,name + " `ASC");

return crplaylists;

}

Note the different uri's MediaStore.Audio.Media for tracks, MediaStore.Audio.Albums for album detail, MediaStore.Audio.Playlists for playlists.

The last parameter in the cr.query is the sort order, My playlist shows an example: name + " `ASC" which sorts by playlist name. I hope this helps

I have developed an app which allows you to manage/create playlists and uses these techniques.<https://play.google.com/store/apps/details?id=com.flyingdutchman.playlistmanager&hl=en> or search Google Play for Playlist Manager.

Android provides a way to register different type of media, such as audio, video, and images, for consumption by any app. This is convenient if your app is, say, a music player or an image editor. Android’s [MediaStore](http://developer.android.com/reference/android/provider/MediaStore.html) is the provider for this meta data, and includes information about the media such as title, artist, size, and location.

If your application does any sort of media content creation, such as image editing or downloading audio from an external website, then you generally want to make that content accessible from any other apps that can consume it. When you create a file you can use the [MediaScannerConnection](http://developer.android.com/reference/android/media/MediaScannerConnection.html) to add the file and its metadata to the MediaStore.

If you delete the file from the file system, the metadata remains in the MediaStore until Android scans the system for new media, which typically happens when the system first boots up or can be called explicitly called in such a way:

sendBroadcast(

new Intent(

Intent.ACTION\_MEDIA\_MOUNTED,

Uri.parse("file://" + Environment.getExternalStorageDirectory())

)

);

While this method works, it is time and resource consuming, as basically the entire file system must be re-scanned. An alternative is to explicitly delete the file from the MediaStore. We’re going to discuss two ways to do this. The first is to query to MediaStore for the content, based on some predicate, and delete based on the unique ID the MediaStore identifies it by. The second, and easier, way to do it is to just specify the predicate in the delete statement. In these example I’m going to be deleting an audio file based on it’s file name and path, but you can easily use this to delete any type of media based on any known information (such as video duration, or image dimensions).

In querying the MediaStore, you should think of it as a SQL database. You need to form your query by specifying the table (the MediaStore’s external content table), the columns you need (the content’s ID), and the where clause (how to identify the content).

To perform the actual query, we’re going to use the [ContentResolver](http://developer.android.com/reference/android/content/ContentResolver.html)’s query() method.

String[] retCol = { MediaStore.Audio.Media.\_ID };

Cursor cur = context.getContentResolver().query(

MediaStore.Audio.Media.EXTERNAL\_CONTENT\_URI,

retCol,

MediaStore.MediaColumns.DATA + "='" + filePath + "'", null, null

);

if (cur.getCount() == 0) {

return;

}

cur.moveToFirst();

int id = cur.getInt(cur.getColumnIndex(MediaStore.MediaColumns.\_ID));

cur.close();

The first argument to query() specifies the columns we want returned, which in this case is only “\_ID”.

The second argument specifies that we want to look at the media stored on the external SD card (which would be internal storage on deices with no SD card).

The third argument is the predicate which specifies what content we’re looking for.

In this case, I’m identifying the file by it’s path in the file system (which is what is stored in the MediaColumns.DATA column).

The fourth and fifth columns are the predicate’s arguments and the ordering, respectively.

I’m including my predicate’s arguments in the predicate itself so that’s not necessary, and if your only looking for one piece of content and your predicate is specific enough to just return one row then the ordering doesn’t matter.

It is very important to make the predicate specific enough so that you’re guaranteed to get the exact ID you’re looking for. In my case I know that there can be only one file at a particular location, but you could use a combination of any columns (such as title, artist, and album) to find the content. Check out the [MediaColumns](http://developer.android.com/reference/android/provider/MediaStore.MediaColumns.html) for all the possibilities.

Once you perform the actual query, you’ll want to check to see whether **the MediaStore** actually contains the content you’re trying to delete.

If you don’t handle this in some way your app will crash while trying to iterate through the cursor.

Once you confirm that the query returned some data, grab the ID by moving the cursor to it’s first position, reading the “\_ID” column, and closing the cursor.

It’s very important that you remember to close the cursor once you’ve finished using it.

Your app won’t crash, but you’ll get memory leaks and complaints in LogCat.

Now that we have the ID that the MediaStore associates with our content, we can call ContentResolver’s delete() method similar to how we called its query() method.

Uri uri = ContentUris.withAppendedId(

MediaStore.Audio.Media.EXTERNAL\_CONTENT\_URI, id

);

context.getContentResolver().delete(uri, null, null);

The delete() method takes 3 arguments: **the Uri** to be deleted, the predicate, and the ***predicate arguments***.

We form the Uri by appending the ID we discovered by querying the MediaStore to the Uri of the audio files on external storage.

Since we know exactly which row we want to delete, we don’t need to specify the predicate or the predicate’s arguments.

The second method to delete the content from the MediaStore takes advantage of the fact that querying and deleting from it are performed almost identically.

context.getContentResolver().delete(

MediaStore.Audio.Media.EXTERNAL\_CONTENT\_URI,

MediaStore.MediaColumns.DATA + "='" + path + "'", null

);

We can use the predicate of the delete() method to specify exactly what we want to delete, rather than having to query for it beforehand.

While this method is more efficient (no extra query, no cursors to deal with), it has **some pitfalls.**

You have no way of explicitly confirming what you’re deleting.

You’re also not able to do advanced queries with this method, **such as if you wanted to delete the most** recently added content (which you could do by ordering the query based on the DATE\_ADDED column).

However, both ways give you a way to confirm what you’ve deleted since the delete() method returns the number of rows that it deleted as an integer.

Topics to learn

* + - * 1. Content Provider

Shared Preferences

SQLite Database

Content Provider

Custom Content Provider

Media Store and Multimedia Techniques

Audio, Video and Images with recipes

Illustrated Example

* https://www.tutorialspoint.com/android/android\_content\_providers.htm
* https://developer.android.com/guide/topics/providers/content-provider-creating.html
* https://developer.android.com/guide/topics/providers/content-provider-basics.html
* https://developer.android.com/guide/topics/providers/content-providers.html
* https://code.tutsplus.com/tutorials/android-fundamentals-working-with-content-providers--mobile-5549
* https://www.androidtutorialpoint.com/storage/android-content-provider-tutorial/
* http://www.grokkingandroid.com/android-tutorial-writing-your-own-content-provider/
* https://guides.codepath.com/android/Creating-Content-Providers
* https://www.sitepoint.com/create-your-own-content-provider-in-android/

**SQLite Database and Content Provider**

* http://www.vogella.com/tutorials/AndroidSQLite/article.html
* https://www.tutorialspoint.com/android/android\_sqlite\_database.htm
* https://developer.android.com/training/basics/data-storage/databases.html
* https://developer.android.com/reference/android/database/sqlite/package-summary.html
* https://developer.android.com/training/basics/data-storage/databases.html#DefineContract
* http://www.androidhive.info/2011/11/android-sqlite-database-tutorial/
* https://www.javatpoint.com/android-sqlite-tutorial
* http://mobilesiri.com/android-sqlite-database-tutorial-using-android-studio/
* https://www.codeproject.com/Articles/1031191/Android-SQLite-for-Beginners

**SharedPreferences in Android**

* https://www.tutorialspoint.com/android/android\_shared\_preferences.htm
* https://developer.android.com/training/basics/data-storage/shared-preferences.html
* http://stackoverflow.com/questions/23024831/android-shared-preferences-example
* http://www.journaldev.com/9412/android-shared-preferences-example-tutorial
* https://www.101apps.co.za/articles/using-androids-sharedpreferences-to-save-data.html
* https://examples.javacodegeeks.com/android/core/content/android-sharedpreferences-example/

**Media Store**

* The Media provider contains meta data for all available media on both internal and external storage devices.
* Android Sound and Media - http://www.vogella.com/tutorials/AndroidMedia/article.html
* http://www.vogella.com/tutorials/AndroidMedia/article.html#supported-formats
* http://www.androiddevelopersolutions.com/2015/12/android-media-store-tutorial-list-all.html
* http://stackoverflow.com/questions/18926633/list-of-albums-from-mediastore-to-adapter-listview
* https://www.sandersdenardi.com/querying-and-removing-media-from-android-mediastore/

**Chapter Three** **Graphical User Interface, Views, Layouts and Dynamic UI**

Objective To understand UI and Views and Custom Views and much more in simple terms

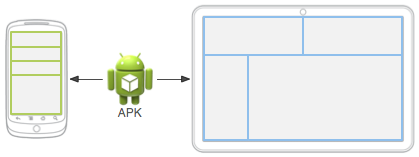
Background

It takes more than just code to build a great app**. Resources** are the additional files and static content that your code uses, such as bitmaps, layout definitions, user interface strings, animation instructions, and more.

You should always externalize resources such as images and strings from your application code, so that you can maintain them independently. Externalizing your resources also allows you to provide alternative resources that support specific device configurations such as different languages or screen sizes, which becomes increasingly important as more Android-powered devices become available with different configurations. In order to provide compatibility with different configurations, you must organize resources in your project's res/directory, using various sub-directories that group resources by type and configuration.



**Figure 1.** Two different devices, each using the default layout (the app provides no alternative layouts).



**Figure 2.** Two different devices, each using a different layout provided for different screen sizes.

For any type of resource, you can specify ***default*** and multiple ***alternative*** resources for your application:

* Default resources are those that should be used regardless of the device configuration or when there are no alternative resources that match the current configuration.
* Alternative resources are those that you've designed for use with a specific configuration. To specify that a group of resources are for a specific configuration, append an appropriate configuration qualifier to the directory name.

For example, while your default UI layout is saved in the **res/layout/** directory, you might specify a different layout to be used when the screen is in landscape orientation, by saving it in the res/layout-land/ directory. Android automatically applies the appropriate resources by matching the device's current configuration to your resource directory names.

Figure 1 illustrates how the system applies the **same layout for two different devices** when there are no alternative resources available.

Figure 2 shows the same application when it **adds an alternative layout resource** for larger screens.

The following documents **provide a complete guide** to how you can organize your application resources, specify alternative resources, access them in your application, and more:

[**Providing Resources**](https://developer.android.com/guide/topics/resources/providing-resources.html)

What kinds of **resources** you can provide in your app, where to save them, and how to create alternative resources for specific device configurations.

[**Accessing Resources**](https://developer.android.com/guide/topics/resources/accessing-resources.html)

How to use the resources you've provided, either by referencing them from your application code or from other XML resources.

[**Handling Runtime Changes**](https://developer.android.com/guide/topics/resources/runtime-changes.html)

How to manage configuration changes that occur while your Activity is running.

[**Localization**](https://developer.android.com/guide/topics/resources/localization.html)

A bottom-up guide to localizing your application using alternative resources. While this is just one specific use of alternative resources, it is very important in order to reach more users.

[**Complex XML Resources**](https://developer.android.com/guide/topics/resources/complex-xml-resources.html)

An XML format for building complex resources like animated vector **drawables** in a single XML file.

[**Resource Types**](https://developer.android.com/guide/topics/resources/available-resources.html)

A reference of various resource types you can provide, describing their XML elements, attributes, and syntax. For example, this reference shows you how to create a resource for application menus, drawables, animations, and more.

Class for accessing an application's resources. This sits on top of the asset manager of the application (accessible through **getAssets())** and provides a high-level API for getting typed data from the assets.

The Android resource system keeps track of all non-code assets associated with an application. You can use this class to access your application's resources. You can generally acquire the **Resources instance** associated with your application with **getResources().**

The Android SDK tools compile your application's resources into the application binary at build time. To use a resource, you **must install it correctly in the source tree** (inside your project's res/ directory) and build your application. As part of the build process, the SDK tools generate symbols for each resource, which you can use in your application code to access the resources.

Using **application resources** makes it easy to update various characteristics of your application without modifying code, and—by providing sets of alternative resources—enables you to optimize your application for a variety of device configurations (such as for different languages and screen sizes). This is an **important** aspect of developing Android applications that are compatible on different types of devices.

Each of the documents in this section describe the usage, format and syntax for a certain type of application resource that you can provide in your resources directory (res/).

Here's a brief summary of each resource type:

[**Animation Resources**](https://developer.android.com/guide/topics/resources/animation-resource.html)

Define pre-determined animations.  
Tween animations are saved in res/anim/ and accessed from the R.anim class.  
Frame animations are saved in res/drawable/ and accessed from the R.drawable class.

[**Color State List Resource**](https://developer.android.com/guide/topics/resources/color-list-resource.html)

Define a color resources that changes based on the View state.  
Saved in res/color/ and accessed from the R.color class.

[**Drawable Resources**](https://developer.android.com/guide/topics/resources/drawable-resource.html)

Define various graphics with bitmaps or XML.  
Saved in res/drawable/ and accessed from the R.drawable class.

[**Layout Resource**](https://developer.android.com/guide/topics/resources/layout-resource.html)

Define the layout for your application UI.  
Saved in res/layout/ and accessed from the R.layout class.

[**Menu Resource**](https://developer.android.com/guide/topics/resources/menu-resource.html)

Define the contents of your application menus.  
Saved in res/menu/ and accessed from the R.menu class.

[**String Resources**](https://developer.android.com/guide/topics/resources/string-resource.html)

Define strings, string arrays, and plurals (and include string formatting and styling).  
Saved in res/values/ and accessed from the R.string, R.array, and R.plurals classes.

[**Style Resource**](https://developer.android.com/guide/topics/resources/style-resource.html)

Define the look and format for UI elements.  
Saved in res/values/ and accessed from the R.style class.

[**More Resource Types**](https://developer.android.com/guide/topics/resources/more-resources.html)

Define values such as booleans, integers, dimensions, colors, and other arrays.  
Saved in res/values/ but each accessed from unique R sub-classes (such as R.bool, R.integer, R.dimen, etc.).

There are many more items which you use to build a good Android application. Apart from coding for the application, you take care **of various other resources like static content** that your code uses, such as bitmaps, colours, layout definitions, user interface strings, animation instructions, and more. These resources are always maintained separately in various sub-directories under **res/** directory of the project.

Here you learn how you can organize your application resources, specify alternative resources and access them in your applications.

## Organize resource in Android Studio

MyProject/

app/

manifest/

AndroidManifest.xml

java/

MyActivity.java

**res/**

drawable/

icon.png

layout/

activity\_main.xml

info.xml

values/

strings.xml

References

* https://developer.android.com/guide/topics/resources/index.html
* https://developer.android.com/guide/topics/resources/overview.html
* channel9.msdn.com/Series/The-Full-Stack/
* https://developer.android.com/guide/topics/resources/available-resources.html
* https://www.tutorialspoint.com/android/android\_resources.htm
* https://www.sitepoint.com/5-resources-for-android-developers/
* https://guides.codepath.com/android/Understanding-App-Resources
* https://code.tutsplus.com/tutorials/android-sdk-app-resources--mobile-20506
* http://www.htmlgoodies.com/beyond/mobile/resources-for-android-app-and-website-developers.html

Your app's **user interface** is everything that the user can see and interact with. Android provides a variety of pre-built UI components such as structured layout objects and UI controls that allow you to build the graphical user interface for your app. Android also provides other UI modules for special interfaces such as dialogs, notifications, and menus.

All user interface elements in an Android app are built using [View](https://developer.android.com/reference/android/view/View.html) and [ViewGroup](https://developer.android.com/reference/android/view/ViewGroup.html) objects. A [View](https://developer.android.com/reference/android/view/View.html) is an object that draws something on the screen that the user can interact with. A [ViewGroup](https://developer.android.com/reference/android/view/ViewGroup.html) is an object that holds other [View](https://developer.android.com/reference/android/view/View.html) (and [ViewGroup](https://developer.android.com/reference/android/view/ViewGroup.html)) objects in order to define the layout of the interface.

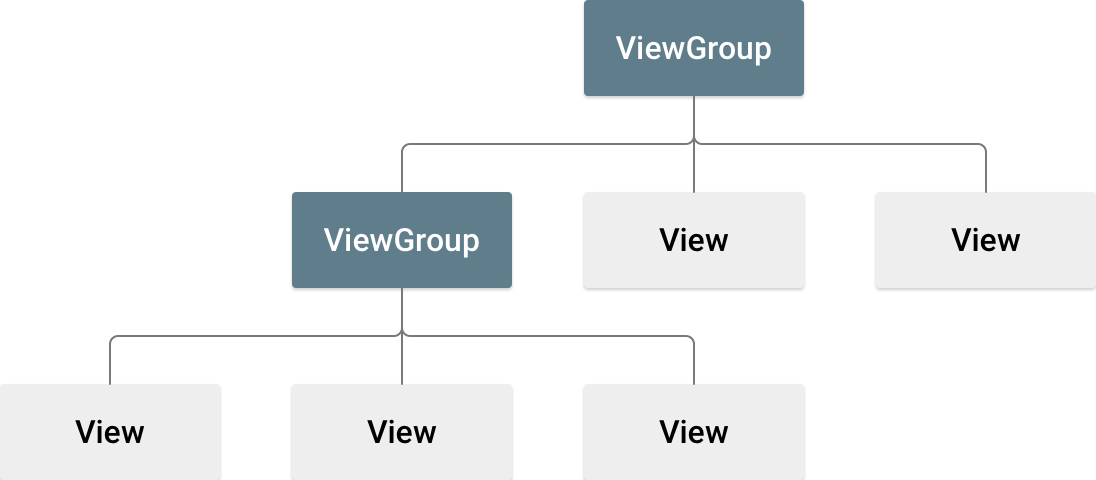
Android provides a collection of both [View](https://developer.android.com/reference/android/view/View.html) and [ViewGroup](https://developer.android.com/reference/android/view/ViewGroup.html) subclasses that offer you common input controls (such as buttons and text fields) and various layout models (such as a linear or relative layout).

## User Interface Layout

The user interface for each component of your app is defined using a hierarchy of [View](https://developer.android.com/reference/android/view/View.html) and [ViewGroup](https://developer.android.com/reference/android/view/ViewGroup.html) objects, as shown in figure 1.

Each view group is an invisible container that organizes child views, while the child views may be input controls or other widgets that draw some part of the UI.

This hierarchy tree can be as simple or complex as you need it to be (but simplicity is best for performance).



**Figure 1.** Illustration of a view hierarchy, which defines a UI layout.

To declare your layout, you can instantiate [View](https://developer.android.com/reference/android/view/View.html) objects in code and start building a tree, but the easiest and most effective way to define your layout is with an XML file. XML offers a human-readable structure for the layout, similar to HTML.

The name of an XML element for a view is respective to the Android class it represents.

So a <TextView> element creates a [TextView](https://developer.android.com/reference/android/widget/TextView.html) widget in your UI, and a <LinearLayout> element creates a [LinearLayout](https://developer.android.com/reference/android/widget/LinearLayout.html) view group.

For **example**, a **simple vertical layout with a text view and a button** looks like this:

<?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="I am a TextView" **/>**  
    <**Button** android:id="@+id/button"  
            android:layout\_width="wrap\_content"  
            android:layout\_height="wrap\_content"  
            android:text="I am a Button" />  
</**LinearLayout**>

When you load a layout resource in your app, Android initializes each node of the layout into a runtime object you can use to define additional behaviors, query the object state, or modify the layout.

For a complete guide to creating a UI layout, see [XML Layouts](https://developer.android.com/guide/topics/ui/declaring-layout.html).

## User Interface Components

You don't have to build all of your UI using [View](https://developer.android.com/reference/android/view/View.html) and [ViewGroup](https://developer.android.com/reference/android/view/ViewGroup.html) objects.

Android provides several app components that offer a standard UI layout for which you simply need to define the content.

These UI components each have a unique set of APIs that are described in their respective documents, such as [Adding the App Bar](https://developer.android.com/training/appbar/index.html), [Dialogs](https://developer.android.com/guide/topics/ui/dialogs.html), and [Status Notifications](https://developer.android.com/guide/topics/ui/notifiers/notifications.html).

An **Android layout** is a class that handles arranging the way its children appear on the screen.  Anything that is a View (or inherits from View) can be a child of a layout. All of the layouts inherit from ViewGroup (which inherits from View) so you can nest layouts.  You could also create your own custom layout by making a class that inherits from ViewGroup.

The standard Layouts are:

* [AbsoluteLayout](http://www.learn-android.com/2010/01/05/android-layout-tutorial/2/)
* [FrameLayout](http://www.learn-android.com/2010/01/05/android-layout-tutorial/3/)
* [LinearLayout](http://www.learn-android.com/2010/01/05/android-layout-tutorial/4/)
* [RelativeLayout](http://www.learn-android.com/2010/01/05/android-layout-tutorial/5/)
* [TableLayout](http://www.learn-android.com/2010/01/05/android-layout-tutorial/6/)

In this article we will examine each of these layouts in detail. I have also created a demo project that uses the code samples from this tutorial and from the [Lots of Lists: Part 1, Simple List Activity tutorial](http://www.learn-android.com/2009/12/05/lots-of-lists-1-simple/). You can download the demo source code [here](http://www.learn-android.com/projects/LayoutDemo.zip). Next: [AbsoluteLayout](http://www.learn-android.com/2010/01/05/android-layout-tutorial/2/)

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

* **Declare UI elements in XML**. Android provides a straightforward XML vocabulary that corresponds to the View classes and subclasses, such as those for widgets and layouts.
* **Instantiate layout elements at runtime**. Your application can create View and ViewGroup objects (and manipulate their properties) programmatically.

The Android framework gives you the flexibility to use either or both of these methods for declaring and managing your application's UI. For example, you could declare your application's default layouts in XML, including the screen elements that will appear in them and their properties. You could then add code in your application that would modify the state of the screen objects, including those declared in XML, at run time.

* You should also try the **Hierarchy Viewer** tool, for debugging layouts — it reveals layout property values, draws wireframes with padding/margin indicators, and full rendered views while you debug on the emulator or device.
* The**layoutopt** tool lets you quickly analyze your layouts and hierarchies for inefficiencies or other problems.

The advantage to declaring your UI in XML is that it enables you to better separate the presentation of your application from the code that controls its behavior. Your UI descriptions are external to your application code, which means that you can modify or adapt it without having to modify your source code and recompile. For example, you can create XML layouts for different screen orientations, different device screen sizes, and different languages.

Additionally, declaring the layout in XML makes it easier to visualize the structure of your UI, so it's easier to debug problems. As such, this document focuses on teaching you how to declare your layout in XML. If you're interested in instantiating View objects at runtime, refer to the **ViewGroup** and **View** class references.

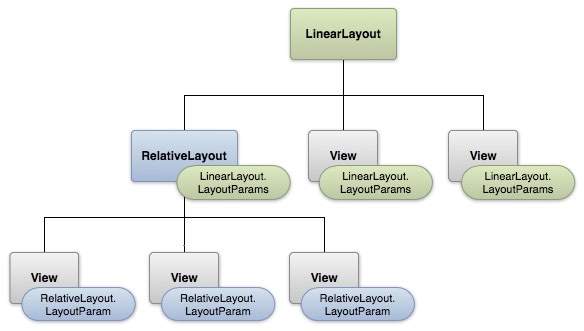
In general, the XML vocabulary for declaring UI elements closely follows the structure and naming of the classes and methods, where element names correspond to class names and attribute names correspond to methods. In fact, the correspondence is often so direct that you can guess what XML attribute corresponds to a class method, or guess what class corresponds to a given XML element. However, note that not all vocabulary is identical. In some cases, there are slight naming differences. For example, the EditText element has a text attribute that corresponds to **EditText.setText().**

**Views in Android**

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.



Every item in a user interface is a subclass of the Android View class (to be precise **android.view.View**). The Android SDK provides a set of pre-built views that can be used to construct a user interface.

Typical examples include standard items such as the Button, CheckBox, ProgressBar and TextView classes. Such views are also referred to as widgets. For requirements that are not met by the widgets supplied with the SDK, new views may be created either by subclassing and extending an existing class, or creating an entirely new component by building directly on top of the View class.

A view can also be comprised of multiple other views (otherwise known as a composite view). Such views are sub-classed from the Android ViewGroup class (**android.view.ViewGroup**) which is itself a subclass of View.

An example of such a view is the RadioGroup, which is intended to contain multiple RadioButton objects such that only one can be in the “on” position at any one time.

In terms of structure, composite views consist of a single parent view (derived from the ViewGroup class and otherwise known as a container view or root element) that is capable of containing other views (known as child views).

Another category of ViewGroup based container view is that of the layout manager.

The Android framework provides several default views. The base class a view is the View. Views are responsible for measuring, layouting and drawing themselves and their child elements (in case of a ViewGroup). Views are also responsible for saving their UI state and handling touch events. Developers can also create custom views and use them in their application.

It is possible to create custom views by:

* Compound views - combining views with a default wiring
* **Custom** views - creating your own views
  + by extending an existing view, e.g. Button
  + by extending the View class

View are typically created to provide a user interface experience with is not possible with the default views. Using custom view allows the developer allow to do certain performance optimization, i.e., in case of a custom layout the development can optimize the layout manager for his use case.

### [How Android draws the view hierarchy](http://www.vogella.com/tutorials/AndroidCustomViews/article.html#how-android-draws-the-view-hierarchy)

Once an activity receives the focus, it must provide the root node of its layout hierarchy to the Android system. Afterwards the Android system starts the drawing procedure.

Drawing begins with the root node of the layout. The layout hierarchy is traversed in the order of declaration, i.e., parents are drawn before their children and children are drawn in the order of declaration.

Drawing ***the layout is a two pass*** process:

* measuring pass - implemented in the`measure(int, int)` method. This happens as a top-down traversal of the view hierarchy. Every view stores its measurements.
* layout pass - implemented in the layout(int, int, int, int) method. This is also a top-down traversal of the view hierarchy. During this phase each layout manager is responsible for positioning all of its children. It uses the sizes computed in the measure pass.

# Creating Custom Views

The Android framework has a large set of **View** classes for interacting with the user and displaying various types of data. But sometimes your app has unique needs that aren’t covered by the built-in views. This class shows you how to create your own views that are robust and reusable.

## Lessons

[**Creating a View Class**](https://developer.android.com/training/custom-views/create-view.html)

Create a class that acts like a built-in view, with custom attributes and support from the Android Studio layout editor.

[**Custom Drawing**](https://developer.android.com/training/custom-views/custom-drawing.html)

Make your view visually distinctive using the Android graphics system.

[**Making the View Interactive**](https://developer.android.com/training/custom-views/making-interactive.html)

Users expect a view to react smoothly and naturally to input gestures. This lesson discusses how to use gesture detection, physics, and animation to give your user interface a professional feel.

[**Optimizing the View**](https://developer.android.com/training/custom-views/optimizing-view.html)

No matter how beautiful your UI is, users won't love it if it doesn't run at a consistently high frame rate. Learn how to avoid common performance problems, and how to use hardware acceleration to make your custom drawings run faster.

# Creating a View Class

A **well-designed custom view** is much like any other well-designed class. It encapsulates a specific set of functionality with an easy to use interface, it uses CPU and memory efficiently, and so forth. In addition to being a well-designed class, though, a custom view should:

* Conform to Android standards
* Provide custom styleable attributes that work with Android XML layouts
* Send accessibility events
* Be compatible with multiple Android platforms.

The Android framework provides a set of base classes and XML tags to help you create a view that meets all of these requirements. This lesson discusses how to use the Android framework to create the core functionality of a view class.

## Subclass a View

All of the view classes defined in the Android framework extend **View**. Your custom view can also extend [View](https://developer.android.com/reference/android/view/View.html) directly, or you can save time by extending one of the existing view subclasses, such as [Button](https://developer.android.com/reference/android/widget/Button.html).

To allow Android Studio to interact with your view, at a minimum you must provide a constructor that takes a [Context](https://developer.android.com/reference/android/content/Context.html) and an **AttributeSe**t object as parameters. This constructor allows the layout editor to create and edit an instance of your view.

class PieChart extends View {  
    public PieChart(Context context, AttributeSet attrs) {  
        super(context, attrs);  
    }  
}

## Define Custom Attributes

To add a built-in **View** to your user interface, you specify it in an XML element and control its appearance and behavior with element attributes. Well-written custom views can also be added and styled via XML. To enable this behavior in your custom view, you must:

* Define custom attributes for your view in a <declare-styleable> resource element
* Specify values for the attributes in your XML layout
* Retrieve attribute values at runtime
* Apply the retrieved attribute values to your view

Topics to learn

* + Managing Application Resources
  + Directories and General Attributes
  + Views and View Groups
    1. Layouts
       1. Frame layout
       2. Table layout
       3. Linear layout
       4. Relative layout
       5. Grid layout
       6. List Activity
          1. XML layout
          2. Row layout
          3. Binding data to list
    2. Attributes of UI elements in common views
       1. UI elements width and height
       2. Margins and Padding
       3. Gravity
    3. Relative layout and layout id
    4. Declaring a layout programmatically
    5. Updating a layout from a separate thread
    6. Manipulation of text
    7. Other Widgets
  + UI Events - Event handlers and listeners
    1. Intercepting the Physical Key Press
    2. Building Menus
    3. Define Menus in XML
    4. Use the Search Key
    5. Reacting to Touch Events
    6. Listen to Fling Gestures
    7. Use Gestures from user interface libraries
    8. Drawing 3D images
  + Android Custom View, Customize a Button (recipe)
  + Binding Data with the **AdapterView** class
  + Design the **AutoTextComplete** View
  + Menus - the options menu, the context menu and sub menus
  + Handling Pictures and Menus with Views
    1. Display Images with Gallery and Grid Views
    2. Use ImageSwitcher View
  + Action Bar - Create an Action bar, Using **ActionBarSherlock**
  + Notifications, Toasts, Alerts, Dialogs
  + Create Your Own Widget or Custom Widgets
  + Navigation as part of UI Design
  + Prototyping User Interfaces as part of UI Design
  + Android User Experience

Illustrated Example

**User Interface**

* https://developer.android.com/guide/topics/ui/index.html
* https://developer.android.com/guide/topics/ui/overview.html
* http://eagle.phys.utk.edu/guidry/android/androidUserInterface.html

**Simple UI, UI Overview, UI**

* https://developer.android.com/training/basics/firstapp/building-ui.html
* https://developer.android.com/guide/topics/ui/overview.html#UIComponents
* https://developer.android.com/guide/topics/ui/index.html

**UI Controls**

* https://www.tutorialspoint.com/android/android\_user\_interface\_controls.htm

**UI Design**

* https://www.tutorialspoint.com/android/android\_ui\_design.htm
* https://www3.ntu.edu.sg/home/ehchua/programming/android/Android\_BasicsUI.html
* https://developer.android.com/training/best-ui.html

**Layouts**

* http://www.learn-android.com/2010/01/05/android-layout-tutorial/
* https://developer.android.com/guide/topics/ui/declaring-layout.html
* https://www.tutorialspoint.com/android/android\_user\_interface\_layouts.htm

**View, ViewGroup**

* http://tutorials.jenkov.com/android/view-viewgroup.html
* https://code.tutsplus.com/tutorials/android-from-scratch-understanding-views-and-view-groups--cms-26043
* https://developer.android.com/reference/android/view/ViewGroup.html
* http://www.techotopia.com/index.php/Understanding\_Android\_Views,\_View\_Groups\_and\_Layouts

**Custom Views, Compound ViewGroup, Custom ViewGroup**

* https://medium.com/@aballano/android-layouts-to-the-next-level-custom-views-compound-viewgroups-and-custom-viewgroups-7a6c3962358d
* http://stacktips.com/tutorials/android/how-to-create-custom-layout-in-android-by-extending-viewgroup-class
* https://developer.android.com/training/custom-views/index.html
* https://developer.android.com/training/custom-views/create-view.html
* http://www.vogella.com/tutorials/AndroidCustomViews/article.html
* https://www.intertech.com/Blog/android-custom-view-tutorial-part-1-combining-existing-views/
* https://code.tutsplus.com/tutorials/android-sdk-creating-custom-views--mobile-14548
* http://stacktips.com/tutorials/android/creating-custom-views-in-android-tutorial
* https://www.101apps.co.za/articles/creating-custom-views.html
* https://www.protechtraining.com/blog/post/816?ncr=1
* https://medium.com/@aballano/android-layouts-to-the-next-level-custom-views-compound-viewgroups-and-custom-viewgroups-7a6c3962358d

**Chapter Four Part I Android Services, Android IPC & RPC, AIDL, APIs**

Objective

Background

A **service** is a component that runs in the background to perform long-running operations without needing to interact with the user and it works even if application is destroyed

A **Service** is an application component that can perform long-running operations in the background, and it does not provide a user interface. Another application component can start a service, and it continues to run in the background even if the user switches to another application. Additionally, a component can bind to a service to interact with it and even perform interprocess communication (IPC). For example, a service can handle network transactions, play music, perform file I/O, or interact with a content provider, all from the background.

These are the three different types of services:

**Scheduled**

A service is *scheduled* when an API such as the [JobScheduler](https://developer.android.com/reference/android/app/job/JobScheduler.html), introduced in Android 5.0 (API level 21), launches the service. You can use the [JobScheduler](https://developer.android.com/reference/android/app/job/JobScheduler.html) by registering jobs and specifying their requirements for network and timing. The system then gracefully schedules the jobs for execution at the appropriate times. The [JobScheduler](https://developer.android.com/reference/android/app/job/JobScheduler.html) provides many methods to define service-execution conditions.

**Note:** If your app targets Android 5.0 (API level 21), Google recommends that you use the [JobScheduler](https://developer.android.com/reference/android/app/job/JobScheduler.html) to execute background services. For more information about using this class, see the [JobScheduler](https://developer.android.com/reference/android/app/job/JobScheduler.html) reference documentation.

**Started**

A service is *started* when an application component (such as an activity) calls [startService()](https://developer.android.com/reference/android/content/Context.html#startService(android.content.Intent)). After it's started, a service can run in the background indefinitely, even if the component that started it is destroyed. Usually, a started service performs a single operation and does not return a result to the caller. For example, it can download or upload a file over the network. When the operation is complete, the service should stop itself.

**Bound**

A service is *bound* when an application component binds to it by calling [bindService()](https://developer.android.com/reference/android/content/Context.html#bindService(android.content.Intent, android.content.ServiceConnection, int)). A bound service offers a client-server interface that allows components to interact with the service, send requests, receive results, and even do so across processes with interprocess communication (IPC). A bound service runs only as long as another application component is bound to it. Multiple components can bind to the service at once, but when all of them unbind, the service is destroyed.

Although this documentation generally discusses started and bound services separately, your service can work both ways—it ***can be started (to run indefinitely) and also allow binding***. It's simply a matter of whether you implement a couple of callback methods: [onStartCommand()](https://developer.android.com/reference/android/app/Service.html#onStartCommand(android.content.Intent, int, int)) to allow components to start it and [onBind()](https://developer.android.com/reference/android/app/Service.html#onBind(android.content.Intent)) to allow binding.

Regardless of whether your application is started, bound, or both, any application component can use the service (even from a separate application) in the same way that any component can use an activity—by starting it with an [Intent](https://developer.android.com/reference/android/content/Intent.html). However, ***you can declare the service as private in the manifest file and block access from other*** applications. This is discussed more in the section about [Declaring the service in the manifest](https://developer.android.com/guide/components/services.html#Declaring).

To create a service, you must create a subclass of [Service](https://developer.android.com/reference/android/app/Service.html) or use one of its existing subclasses. In your implementation, you must override some callback methods that handle key aspects of the service lifecycle and provide a mechanism that allows the components to bind to the service, if appropriate. These are the most important callback methods that you should override:

[**onStartCommand()**](https://developer.android.com/reference/android/app/Service.html#onStartCommand(android.content.Intent, int, int))

The system invokes this method by calling [startService()](https://developer.android.com/reference/android/content/Context.html#startService(android.content.Intent)) when another component (such as an activity) requests that the service be started. When this method executes, the service is started and can run in the background indefinitely. If you implement this, it is your responsibility to stop the service when its work is complete by calling [stopSelf()](https://developer.android.com/reference/android/app/Service.html#stopSelf()) or [stopService()](https://developer.android.com/reference/android/content/Context.html#stopService(android.content.Intent)). If you only want to provide binding, you don't need to implement this method.

[**onBind()**](https://developer.android.com/reference/android/app/Service.html#onBind(android.content.Intent))

The system invokes this method by calling [bindService()](https://developer.android.com/reference/android/content/Context.html#bindService(android.content.Intent, android.content.ServiceConnection, int)) when another component wants to bind with the service (such as to perform RPC). In your implementation of this method, you must provide an interface that clients use to communicate with the service by returning an [IBinder](https://developer.android.com/reference/android/os/IBinder.html). You must always implement this method; however, if you don't want to allow binding, you should return null.

[**onCreate()**](https://developer.android.com/reference/android/app/Service.html#onCreate())

The system invokes this method to perform one-time setup procedures when the service is initially created (before it calls either [onStartCommand()](https://developer.android.com/reference/android/app/Service.html#onStartCommand(android.content.Intent, int, int)) or [onBind()](https://developer.android.com/reference/android/app/Service.html#onBind(android.content.Intent))). If the service is already running, this method is not called.

[**onDestroy()**](https://developer.android.com/reference/android/app/Service.html#onDestroy())

The system invokes this method when the service is no longer used and is being destroyed. Your service should implement this to clean up any resources such as threads, registered listeners, or receivers. This is the last call that the service receives.

If a component starts the service by calling [startService()](https://developer.android.com/reference/android/content/Context.html#startService(android.content.Intent)) (which results in a call to [onStartCommand()](https://developer.android.com/reference/android/app/Service.html#onStartCommand(android.content.Intent, int, int))), the service continues to run until it stops itself with [stopSelf()](https://developer.android.com/reference/android/app/Service.html#stopSelf()) or another component stops it by calling [stopService()](https://developer.android.com/reference/android/content/Context.html#stopService(android.content.Intent)).

If a component calls [bindService()](https://developer.android.com/reference/android/content/Context.html#bindService(android.content.Intent, android.content.ServiceConnection, int)) to create the service and [onStartCommand()](https://developer.android.com/reference/android/app/Service.html#onStartCommand(android.content.Intent, int, int)) is *not* called, the service runs only as long as the component is bound to it. After the service is unbound from all of its clients, the system destroys it.

The Android system force-stops a service only when memory is low and it must recover system resources for the activity that has user focus. If the service is bound to an activity that has user focus, it's less likely to be killed; if the service is declared to [run in the foreground](https://developer.android.com/guide/components/services.html#Foreground), it's rarely killed. If the service is started and is long-running, the system lowers its position in the list of background tasks over time, and the service becomes highly susceptible to killing—if your service is started, you must design it to gracefully handle restarts by the system. If the system kills your service, it restarts it as soon as resources become available, but this also depends on the value that you return from [onStartCommand()](https://developer.android.com/reference/android/app/Service.html#onStartCommand(android.content.Intent, int, int)). For more information about when the system might destroy a service, see the [Processes and Threading](https://developer.android.com/guide/components/processes-and-threads.html) document.

Topics to learn

* + Types of Services
  + Services and Threads
  + Android IPC and RPC - recipe
  + Android Interface Definition Language (AIDL) and its Uses
  + The APIs of Android

Illustrated Example

* https://developer.android.com/guide/components/services.html
* https://www.tutorialspoint.com/android/android\_services.htm
* https://www.javatpoint.com/android-service-tutorial
* http://www.vogella.com/tutorials/AndroidServices/article.html
* http://blog.rhesoft.com/2016/03/14/tutorial-**developing-android-background-services**-android-studio/
* http://stacktips.com/tutorials/android/android-service-example
* http://www.techotopia.com/index.php**/Implementing\_an\_Android\_Started\_Service**\_%E2%80%93\_A\_Worked\_Example
* http://inchoo.net/dev-talk/android-development/android-simple-service/

Chapter Four Part II **AsyncTask, JobScheduler, Loader, Handler**

Background

**AsyncTask** enables proper and easy use of the UI thread. This class allows you to perform background operations and publish results on the UI thread without having to manipulate threads and/or handlers.

AsyncTask is **designed to be a helper class** around [Thread](https://developer.android.com/reference/java/lang/Thread.html) and [Handler](https://developer.android.com/reference/android/os/Handler.html) and does not constitute a generic threading framework. **AsyncTasks** should ideally be used for short operations (a few seconds at the most.) If you need to keep threads running for long periods of time, it is highly recommended you use the various APIs provided by the **java.util.concurrent** package such as [Executor](https://developer.android.com/reference/java/util/concurrent/Executor.html), [ThreadPoolExecutor](https://developer.android.com/reference/java/util/concurrent/ThreadPoolExecutor.html) and [FutureTask](https://developer.android.com/reference/java/util/concurrent/FutureTask.html).

An asynchronous task is defined by a computation that runs on a background thread and whose result is published on the UI thread. An asynchronous task is defined by 3 generic types, called Params, Progress and Result, and 4 steps, called onPreExecute, doInBackground, onProgressUpdate and onPostExecute.

**The JobScheduler class** resides in the **android.app.job** namespace and the API provides an interface for scheduling jobs that execute in the application's own process in the background.

This means that the jobs do not run as part of the system; instead, they run in the context of the application.

By default, any network calls in an Android application are done ASAP and use battery power. However, there are certain scenarios that are good candidates for background tasks because there is no urgency for them and *does not require using precious resources* like battery and network radio.

* **Uploading user data**: When the application involved uploading user data that is not time-sensitive.
* Downloading content: When the application can download information that is not urgent; for example, refresh a TV programming guide.

In such situations, using the JobScheduler API can help. Candidate jobs for JobScheduler need to be self-contained; in other words application behavior should not be impacted if the task cannot be completed instantaneously and in one go.

**Prior to the availability** of the JobScheduler class, app developers had use Handlers and AlarmManagers, but both of them were limited in ability. Handlers and AlarmManagers did not provide persistence, and did not have a good failure handling story. App developers needed to work with SyncManager to avoid such issues.

Now, with the JobScheduler class, all that developer pain is gone.

## What's Different with JobScheduler

**JobScheduler** supports a couple of modes that can help with scheduling jobs.

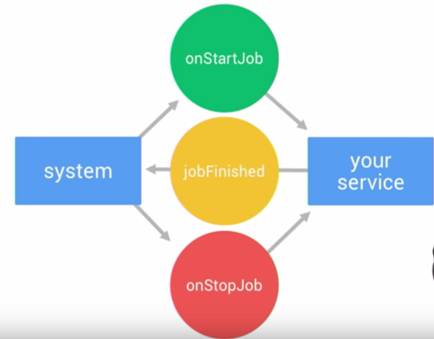
* **Device-idle mode:** This is the time when the device is idle and is in charging mode.
* **Network activity aware:** When the device is connected to Internet over WiFi instead of cellular radio.

## How to Set Up the Job

We need to specify the constraints in which the job should execute. Example of constraints are:

* Run this when the phone is plugged in
* Run this when on an unmetered connection, but only run for one hour
* Wait at least 10 minutes to start this job initially, and then whenever the device is idle
* Every 'x' minutes, whenever network is available let me (app) know so that I (app) can do a small health check.

**Schedule**r only runs the job when all the criteria are satisfied. The only exception is when you specify a deadline. When a defined deadline expires, it overwrites the constraints to execute the job. Because of the encapsulation that Android system provides, app developers need to implement only **two** methods. When the job is finished, the task needs to call **jobFinished** to indicate that the job is complete.

  
**Figure 1:** Android's encapsulation features

Topics to learn

* + - * 1. Background tasks

Handlers and Message Queues

Async Tasks

Loaders

JobScheduler

Illustrated Example

* https://developer.android.com/training/best-background.html
* https://developer.android.com/reference/android/os/AsyncTask.html
* http://www.developer.com/ws/android/programming/executing-**long-running-background-tasks**-in-android-apps-without-an-interface-using-services.html

**Chapter Five Android Threading, Multiple Threads, Thread Lifecycle**

Objective All about threads in Java, Android and multiple threads

Background

When an application component starts and the application does not have any other components running, the Android system starts a new Linux process for the application with a single thread of execution. By default, all components of the same application run in the same process and thread (called the "main" thread). If an application component starts and there already exists a process for that application (because another component from the application exists), then the component is started within that process and uses the same thread of execution. However, you can arrange for different components in your application to run in separate processes, and you can create additional threads for any process.

## Processes

By default, all components of the same application run in the same process and most applications should not change this. However, if you find that you need to control which process a certain component belongs to, you can do so in the manifest file.

The manifest entry for each type of component element—[<activity>](https://developer.android.com/guide/topics/manifest/activity-element.html), [<service>](https://developer.android.com/guide/topics/manifest/service-element.html), [<receiver>](https://developer.android.com/guide/topics/manifest/receiver-element.html), and [<provider>](https://developer.android.com/guide/topics/manifest/provider-element.html)—supports an android:process attribute that can specify a process in which that component should run. You can set this attribute so that each component runs in its own process or so that some components share a process while others do not. You can also set android:process so that components of different applications run in the same process—provided that the applications share the same Linux user ID and are signed with the same certificates.

The [<application>](https://developer.android.com/guide/topics/manifest/application-element.html) element also supports an android:process attribute, to set a default value that applies to all components.

Android might decide to shut down a process at some point, when memory is low and required by other processes that are more immediately serving the user. Application components running in the process that's killed are consequently destroyed. A process is started again for those components when there's again work for them to do.

When deciding which processes to kill, the Android system weighs their relative importance to the user. For example, it more readily shuts down a process hosting activities that are no longer visible on screen, compared to a process hosting visible activities. The decision whether to terminate a process, therefore, depends on the state of the components running in that process.

The details of the process lifecycle and its relationship to application states is discussed in Processes and Application Lifecycle.

## Threads

When an application is launched, the system creates a thread of execution for the application, called "main." This thread is very important because it is in charge of dispatching events to the appropriate user interface widgets, including drawing events. It is also the thread in which your application interacts with components from the Android UI toolkit (components from the **android.widget** and **android.view** packages). As such, the main thread is also sometimes called the UI thread.

The system does not create a separate thread for each instance of a component. All components that run in the same process are instantiated in the UI thread, and system calls to each component are dispatched from that thread. Consequently, methods that respond to system callbacks (such as **onKeyDown()** to report user actions or a lifecycle callback method) always run in the UI thread of the process.

For instance, when the user touches a button on the screen, your app's UI thread dispatches the touch event to the widget, which in turn sets its pressed state and posts an invalidate request to the event queue. The UI thread **dequeues the request and notifies the widget** that it should redraw itself.

When your app performs intensive work in response to user interaction, this single thread model can yield poor performance unless you implement your application properly. Specifically, if everything is happening in the UI thread, performing long operations such as network access or database queries will block the whole UI. When the thread is blocked, no events can be dispatched, including drawing events. From the user's perspective, the application appears to hang. Even worse, if the UI thread is blocked for more than a few seconds (about 5 seconds currently) the user is presented with the infamous "[application not responding](http://developer.android.com/guide/practices/responsiveness.html)" (ANR) dialog. The user might then decide to quit your application and uninstall it if they are unhappy.

Additionally, the Android UI toolkit is not thread-safe. So, you must not manipulate your UI from a worker thread—you must do all manipulation to your user interface from the UI thread. Thus, there are simply two rules to Android's single thread model:

1. Do not block the UI thread
2. Do not access the Android UI toolkit from outside the UI thread

### Worker threads

Because of the single threaded model described above, it's vital to the responsiveness of your application's UI that you do not block the UI thread. If you have operations to perform that are not instantaneous, you should make sure to do them in separate threads ("background" or "worker" threads).

However, note that you cannot update the UI from any thread other than the UI thread or the "main" thread.

## The Application Main Thread

When an Android application is first started, the runtime system creates a single thread in which all application components will run by default. This thread is generally referred to as the main thread. The primary role of the main thread is to handle the user interface in terms of event handling and interaction with views in the user interface. Any additional components that are started within the application will, by default, also run on the main thread.

Any component within an application that performs a time consuming task using the main thread will cause the entire application to appear to lock up until the task is completed. This will typically result in the operating system displaying an “Application is unresponsive” warning to the user. Clearly, this is far from the desired behavior for any application. In such a situation, this can be avoided simply by launching the task to be performed in a separate thread, allowing the main thread to continue unhindered with other tasks.

## Thread Handlers

Clearly one of the key rules of application development is never to perform time-consuming operations on the main thread of an application. The second, equally important rule is that the code within a separate thread must never, under any circumstances, directly update any aspect of the user interface. Any changes to the user interface must always be performed from within the main thread. The reason for this is that the Android UI toolkit is **not** thread-safe. Attempts to work with non thread-safe code from within multiple threads will typically result in intermittent problems and unpredictable application behavior.

In the event that the code executing in a thread needs to interact with the user interface, it must do so by synchronizing with the main UI thread. This is achieved by creating a handler within the main thread, which, in turn, receives messages from another thread and updates the user interface accordingly.

Topics to learn

* + Threading in Linux, Java
  + Threading in Android
    1. UI Thread
    2. Binder Threads
    3. Background Threads
    4. Linux Process and Threads
    5. Scheduling
  + Thread Communication
    1. Pipes
    2. Shared Memory
    3. Blocking Queue
    4. Android Messaging Passing
    5. Communicating with the UI Thread
  + Interprocess Communication - IPC and RPC
  + Memory Management
  + Managing the Lifecycle of a Basic Thread
  + Handler Thread - a high level queuing mechanism
  + Control over Thread Execution through the Executor Framework
  + Tying a Background Task to the UI Thread with AsyncTask
  + Services
  + Access Content Providers with AsyncQueryHandler
  + Automatic background execution with Loaders
  + Using a CusorLoader
  + Implement a RPC, use messsengers, use a resultreceiver
  + Selecting an Asynchronous Technique

Illustrated Example

* https://news.realm.io/news/android-threading-background-tasks/?gclid=COiS\_bDC2tMCFdaHaAodRIcAIQ
* http://www.vogella.com/tutorials/**AndroidBackgroundProcessing**/article.html with handlers, AsyncTask and loaders
* https://developer.android.com/guide/components/processes-and-threads.html
* https://developer.android.com/training/multiple-threads/define-runnable.html - specify the code to run on the thread
* http://www.techotopia.com/index.php/A\_Basic\_Overview\_of\_Android\_Threads\_and\_Thread\_handlers

**Chapter Six Hardware Interface and Sensors**

Objective To learn about the different components that are used in an Android device and its sensors

Background

The **android.hardware** class provides support for hardware features, such as the camera and other sensors. Be aware that not all Android-powered devices support all hardware features, so you should declare hardware that your application requires using the <uses-feature> manifest element.

### DIYers, let's make a complete list of Android Hardware Interface Tools and Solutions

These tools and solutions are among the most popular ways to interface an Android device with the physical/eletronics/mechanic world. But sure, you can help us to include more if you see something else:)

1. **Bluetooth** module+Arduino+Android Application from Cellbots :  
<http://code.google.com/p/cellbots/>  
2. Probably the easiest way to start your first DIY Phone Gadgets Project is to use infrared signal. You can purchase a cheap IRDroid module here:  
<http://www.irdroid.com/purchase/?ap_id=1004>  
3. IOIO for Android:  
<http://ytai-mer.blogspot.com/2011/04/meet-ioio-io-for-android.html>  
4. UDOO: the BEST solution so far to combine Android and Ardunio seemlessly.  
<http://www.diyphonegadgets.com/2013/10/udoo-best-tool-so-far-to-create-diy.html>  
  
5. Arduino + USB Host Shield from Microbridge project:  
<http://code.google.com/p/microbridge/>  
6. **PIC24 board** from Microbridge PIC project:  
<http://code.google.com/p/microbridge-pic/>  
7. **Google ADK** (Android Open Accessory Development Kit):  
<http://developer.android.com/guide/topics/usb/adk.html>  
8. 3.5mm **Stereo Audio Cable** from TRRSTAN cellphone robot:  
<http://www.allthingsgeek.com/>

9. **Bluetooth+Arduino+Android** Library from Amarino:  
[http://www.amarino-toolkit.net/](http://www.google.com/url?sa=D&q=http://www.amarino-toolkit.net/&usg=AFQjCNG-FOL0-0lFtFu2ir-vMJDEigdvnw)  
10. The **PhoneDrone Board** (Android to PPM RC interface board from DIY  
Drones):  
<http://diydrones.com/profiles/blogs/new-diy-drones-product-preview>  
11. **Android ADK** with a standard Arduino Uno and USB Host Shield:  
<http://marioboehmer.blogspot.com/2011/05/android-adk-with-standard-arduino-uno.html>  
  
12. **Native USB** host support (Android 3.1 or above) to control simple USB peripherals:  
<http://developer.android.com/guide/topics/usb/host.html>  
13. IOIO+WiFi dongle. It's a project at its very beginning. IOIO's creator Ytai has just finished [adding Bluetooth to IOIO](http://www.diyphonegadgets.com/2011/10/ioio-over-bluetooth-released-officially.html), and expecting experienced people helping him writing WiFi dongle drivers. Please contact him if you have a clue.  
14**. Android+Raspberry Pi+WiFi** dongle. It's really amazing that this tiny pen-sized Linux PC costs only 25 US dollars and it runs Linux! The combination of Raspberry Pi's size, price and performance makes it a nice candidate of our DIY gadget brain, providing diverse interactions with Android. [http://www.raspberrypi.org](http://www.raspberrypi.org/)  
15. The Electric Sheep board from Sparkfun, a development tool (similar to the Arduino Mega ADK) for creating custom Android accessories. <http://www.sparkfun.com/products/10745>

***Camera in Android Device***

These are the following two ways, in which you can use camera in your application

* Using existing android camera application in our application
* Directly using Camera API provided by android in our application

## Understanding basic classes of Camera Intent and API

There are mainly four classes that we are going to discuss.

#### Intent

By the help of 2 constants of **MediaStore** class, we can capture picture and video without using the instance of Camera class.

1. ACTION\_IMAGE\_CAPTURE
2. ACTION\_VIDEO\_CAPTURE

#### Camera

It is main class of camera api, that can be used to take picture and video.

#### SurfaceView

It represents a surface view or preview of live camera.

#### MediaRecorder

It is used to record video using camera. It can also be used to record audio files as we have seen in the previous example of media framework.

**Camera API** - The Android framework includes support for various cameras and camera features available on devices, allowing you to capture pictures and videos in your applications. This document discusses a quick, simple approach to image and video capture and outlines an advanced approach for creating custom camera experiences for your users.

***Telephony and*** TelephonyManager and the public class TelephonyManager extends [Object](https://developer.android.com/reference/java/lang/Object.html)

|  |  |
| --- | --- |
| [java.lang.Object](https://developer.android.com/reference/java/lang/Object.html) | |
| ↳ | android.telephony.TelephonyManager |

Provides access to information about the telephony services on the device. Applications can use the methods in this class to determine telephony services and states, as well as to access some types of subscriber information.

Applications can also register a listener to receive notification of telephony state changes. You do not instantiate this class directly; instead, you retrieve a reference to an instance through [Context.getSystemService(Context.TELEPHONY\_SERVICE)](https://developer.android.com/reference/android/content/Context.html#getSystemService(java.lang.Class<T>)).

The returned TelephonyManager will use the default subscription for all calls. To call an API for a specific subscription, use [createForSubscriptionId(int)](https://developer.android.com/reference/android/telephony/TelephonyManager.html#createForSubscriptionId(int)).

e.g. telephonyManager = defaultSubTelephonyManager.createForSubscriptionId(subId);

Note that access to some telephony information is permission-protected. Your application cannot access the protected information unless it has the appropriate permissions declared in its manifest file. Where permissions apply, they are noted in the methods through which you access the protected information.

---------------------------------------------------------------------------------------------------------------------------------------

***Bluetooth/ Bluetooth network stack***

The Android platform includes support for **the Bluetooth network stack**, which allows a device to wirelessly exchange data with other Bluetooth devices. The application framework provides access to the Bluetooth functionality through the Android Bluetooth APIs. These APIs let applications wirelessly connect to other Bluetooth devices, enabling point-to-point and multipoint wireless features.

Using the Bluetooth APIs, an Android application can perform the following:

* Scan for other Bluetooth devices
* Query the local Bluetooth adapter for paired Bluetooth devices
* Establish RFCOMM channels
* Connect to other devices through service discovery
* Transfer data to and from other devices
* Manage multiple connections

This page focuses on *Classic Bluetooth*. Classic Bluetooth is the right choice for more battery-intensive operations, which include streaming and communicating between Android devices. For Bluetooth devices with low power requirements, Android 4.3 (API level 18) introduces API support for Bluetooth Low Energy. To learn more, see [Bluetooth Low Energy](https://developer.android.com/guide/topics/connectivity/bluetooth-le.html).

This document describes different Bluetooth profiles, including the Health Device Profile. It then explains how to use the Android Bluetooth APIs to accomplish the four major tasks necessary to communicate using Bluetooth: setting up Bluetooth, finding devices that are either paired or available in the local area, connecting devices, and transferring data between devices.

Among many ways, **Bluetooth** is a way to send or receive data between two different devices. Android platform includes support for the Bluetooth framework that allows a device to wirelessly exchange data with other Bluetooth devices.

Android provides **Bluetooth API** to perform these different operations.

* Scan for other Bluetooth devices
* Get a list of paired devices
* Connect to other devices through service discovery

Android provides **BluetoothAdapter** class to communicate with Bluetooth. Create an object of this calling by calling the static method **getDefaultAdapte**r().

---------------------------------------------------------------------------------------------------------------------------------------

***Near Field Communication (NFC)***

NFC stands for **Near Field Communication**, and as the name implies it provides a wireless communication mechanism between two compatible devices. NFC is a short range wireless technology having a range of 4cm or less for two devices to share data.

## How It Works

Like Bluetooth and WiFi, and all manner of other wireless signals, NFC works on the principle of sending information over radio waves. Through NFC data is send through electromagnetic induction between two devices.

NFC works on the bases of tags , it allows you to share some amount of data between an NFC tag and an android powered device or between two android powered devices. Tags have various set of complexities. The Data stored in the tag can be written in a variety of formats, but android APIs are based around a NFC standard called as **NFC Data Exchange Format(NDEF).**.

The transmission frequency for data across NFC is 13.56 megahertz, and data can be sent at either 106, 212 or 424 kilobits per second, which is quick enough for a range of data transfers from contact details to swapping pictures, songs and videos.

Android powered devices with NFC supports following three main modes of operations −

### Three Modes of Operation

* **Reader/Writer Mode** − It allows the NFC device to read or write passive NFC tags.
* **P2P mode** − This mode allows NFC device to exchange data with other NFC peers.
* **Card emulation mode** − It allows the NFC device itself to act as an NFC card, so it can be accessed by an external NFC reader.

This document describes the **basic NFC tasks** you perform in Android. It explains how to send and receive NFC data in the form of NDEF messages and describes the Android framework APIs that support these features. For more advanced topics, including a discussion of working with non-NDEF data, see **Advanced NFC.**

There are **two major uses cases** when working with **NDEF data** and Android:

* Reading NDEF data from an NFC tag
* Beaming NDEF messages from one device to another with [**Android Beam**™](https://developer.android.com/guide/topics/connectivity/nfc/nfc.html#p2p)

Reading NDEF data from an NFC tag is handled with the [tag dispatch system](https://developer.android.com/guide/topics/connectivity/nfc/nfc.html#tag-dispatch), which analyzes discovered NFC tags, appropriately categorizes the data, and starts an application that is interested in the categorized data. An application that wants to handle the scanned NFC tag can [declare an intent filter](https://developer.android.com/guide/topics/connectivity/nfc/nfc.html#filtering-intents) and request to handle the data.

The **Android Beam™** feature allows a device to push an NDEF message onto another device by physically tapping the devices together. This interaction provides an easier way to send data than other wireless technologies like Bluetooth, because with NFC, no manual device discovery or pairing is required. The connection is automatically started when two devices come into range. Android Beam is available through a set of NFC APIs, so any application can transmit information between devices. For example, the Contacts, Browser, and YouTube applications use Android Beam to share contacts, web pages, and videos with other devices.

When we develop **an Android NFC Tag**, the first thing we want is our app is notified when we get near a NFC tag. To this purpose we use a [intent filter](http://developer.android.com/guide/components/intents-filters.html). Android SDK provides three different filter that we can use with different level of priority:

* *ACTION\_NDEF\_DISCOVERED*
* *ACTION\_TECH\_DISCOVERED*
* *ACTION\_TAG\_DISCOVERED*

---------------------------------------------------------------------------------------------------------------------------------------

***USB / USB Host -*** When your Android-powered device is in USB host mode, it acts as the USB host, powers the bus, and enumerates connected USB devices. USB host mode is supported in Android 3.1 and higher.

## API Overview - Before you begin, it is important to understand the classes that you need to work with. The following table describes the USB host APIs in the android.hardware.usb package. A class representing an interface on a UsbDevice. USB devices can have one or more interfaces, each one providing a different piece of functionality, separate from the other interfaces. An interface will have one or more UsbEndpoints, which are the channels by which the host transfers data with the device. When it comes to USB, an Android device can operate in two modes—USB device mode or USB host mode.

--------------------------------------------------------------------------------------------------------------------------------------

***Sensors***

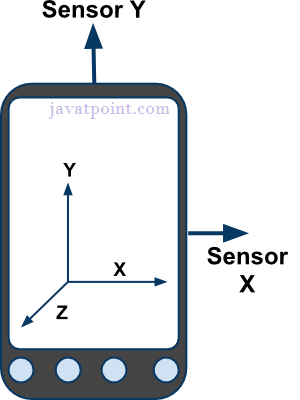
Most of the android devices have built-in sensors that measure motion, orientation, and various environmental condition. The android platform supports *three broad categories* of sensors.

* Motion Sensors
* Environmental sensors
* Position sensors

Some of the sensors are hardware based and some are software based sensors. Whatever the sensor is, android allows us to get the raw data from these sensors and use it in our application. For this android provides us with some classes.

Android provides SensorManager and Sensor classes to use the sensors in our application. In order to use sensors, first thing you need to do is to instantiate the object of SensorManager class.

**Sensors** can be used to monitor the three-dimensional device movement or change in the environment of the device. Android provides sensor api to work with different types of sensors.



## Types of Sensors

Android supports three types of sensors:

#### 1) Motion Sensors

These are used to measure acceleration forces and rotational forces along with three axes.

#### 2) Position Sensors

These are used to measure the physical position of device.

#### 3) Environmental Sensors

These are used to measure the environmental changes such as temperature, humidity etc.

## Android Sensor API

**Android sensor api provides many classes and interface. The important classes and interfaces of sensor api are as follows:**

#### 1) SensorManager class

**The android.hardware.SensorManager class provides methods :**

* **to get sensor instance,**
* **to access and list sensors,**
* **to register and unregister sensor listeners etc.**

**You can get the instance of SensorManager by calling the method getSystemService() and passing the SENSOR\_SERVICE constant in it.**

1. **SensorManager sm = (SensorManager)getSystemService(SENSOR\_SERVICE);**

#### 2) Sensor class

**The android.hardware.Sensor class provides methods to get information of the sensor such as sensor name, sensor type, sensor resolution, sensor type etc.**

#### 3) SensorEvent class

**Its instance is created by the system. It provides information about the sensor.**

#### 4) SensorEventListener interface

**It provides two call back methods to get information when sensor values (x,y and z) change or sensor accuracy changes.**

|  |  |
| --- | --- |
| **Public and abstract methods** | **Description** |
| **void onAccuracyChanged(Sensor sensor, int accuracy)** | it is called when sensor accuracy is changed. |
| **void onSensorChanged(SensorEvent event)** | it is called when sensor values are changed. |

Most Android-powered devices have built-in sensors **that measure motion, orientation, and various environmental** conditions. These sensors are capable of providing raw data with high precision and accuracy, and are useful if you want to monitor three-dimensional device movement or positioning, or you want to monitor changes in the ambient environment near a device.

Topics to learn

* + Customizing the Camera
  + Gyroscope, Accelerometer - Getting a device's rotational attitude
  + Temperature and Light Sensors - Using these
  + Telephony - use telephony manager, listen for phone states, dial a phone number
  + Bluetooth - turn it on, discover other devices, pairing bonded devices, open a socket, use device vibration, access wireless network
  + NFC - read and write NFC tags
  + USB
  + SMS - single and multipart messages, receive a message, use emulator controls to send a message, use TelephonyManager to obtain device information and Auto-send SMS based on Received SMS

Illustrated Example

# How to develop an app using the Camera Remote API

* https://developer.sony.com/2013/11/29/how-to-develop-an-app-using-the-camera-remote-api-2/

**USB Interface Android Documentation**

* https://developer.android.com/reference/android/hardware/usb/UsbInterface.html

**Android Hardware**

* https://developer.android.com/reference/android/hardware/package-summary.html

**DIY Phone Gadgets**

* http://www.diyphonegadgets.com/2011/10/diyers-lets-make-complete-list-of.html

**Camera**

* https://www.tutorialspoint.com/android/android\_camera.htm
* https://developer.android.com/guide/topics/media/camera.html
* Using an Intent to take a photo - http://www.vogella.com/tutorials/AndroidCamera/article.html
* https://www.javatpoint.com/**android-camera-tutorial**
* http://www.androidhive.info/2013/09/android-working-with-camera-api/
* http://blog.rhesoft.com/2015/04/02/tutorial-how-to-use-camera-with-android-and-android-studio/
* https://inducesmile.com/android/android-camera-api-tutorial/
* https://www.simplifiedcoding.net/android-camera-app-tutorial-create-a-simple-camera-app/
* http://www.journaldev.com/13270/android-capture-image-camera-gallery

**Telephony and TelephonyManager**

* https://developer.android.com/reference/android/telephony/TelephonyManager.html
* https://www.javatpoint.com/android-telephony-manager-tutorial
* https://developer.android.com/reference/android/telephony/TelephonyManager.html
* http://www.learn-android-easily.com/2012/11/using-telephony-manager-in-android.html
* http://stackoverflow.com/questions/7514163/using-telephony-manager-in-android-to-find-imei-number
* http://www.devlper.com/2010/06/using-android-telephonymanager/

**Bluetooth**

* https://developer.android.com/guide/topics/connectivity/bluetooth.html
* https://www.tutorialspoint.com/android/android\_bluetooth.htm
* https://www.javatpoint.com/android-bluetooth-tutorial
* https://code.tutsplus.com/tutorials**/create-a-bluetooth-scanner-with-androids-bluetooth**-api--cms-24084
* http://www.theappguruz.com/blog/android-**bluetooth-connection-demo**
* https://examples.javacodegeeks.com/android/core/bluetooth/bluetoothadapter**/android-bluetooth-example/**
* https://karanbalkar.com/2014/02/tutorial-73-**working-with-bluetooth-in-android/**

**NFC**

* https://www.tutorialspoint.com/android/android\_nfc\_guide.htm
* http://www.survivingwithandroid.com/2015/03/nfc-in-android-ndef-2.html
* https://www.sitepoint.com/learn-**android-nfc-basics-building-a-simple-messenger**/
* http://www.framentos.com/en/android-tutorial/2012/07/31/write-hello-**world-into-a-nfc-tag-with-a**/
* http://www.developer.com/ws/android/**nfc-programming-in-android**.html
* https://www.learn2crack.com/2016/10/**android-reading-and-writing-nfc-tags**.html
* http://stackoverflow.com/questions/5762234/nfc-tutorial**-for-android-other-than-api**-demo
* https://webshop.d-logic.net/**nfc-rfid-device**-comparison?gclid=CIfOwc212tMCFYQEaAod2EMCWg

**USB**

* https://developer.android.com/guide/topics/connectivity/usb/host.html#api
* http://android-er.blogspot.in/2014/02/android-usb-host-mode-step-by-step.html
* http://www.embedded.com/design/connectivity/4437620/The-basics-of-USB-device-development-using-the-Android-frameworkThe-basics-of-USB-device-development-using-the-Android-framework
* http://mobilemerit.com/android-app-for-usb-host-with-source-code/
* http://stackoverflow.com/questions/11638216/how-to-make**-an-basic-android-usb-host**-application
* https://www.b4x.com/android/forum/threads/android-**usb-host-tutorial-adbtest**.11289/
* https://visualgdb.com/tutorials/android/**usbdebug/**
* http://stackoverflow.com/questions/10183794/how**-to-communicate-with-a-usb-device**

**Sensors in Android Device**

* https://www.javatpoint.com/android-sensor-tutorial
* https://www.tutorialspoint.com/android/android\_sensors.htm
* https://developer.android.com/guide/topics/sensors/sensors\_overview.html
* https://developer.android.com/guide/topics/sensors/sensors\_**motion**.html
* http://www.survivingwithandroid.com/2013/09/android-sensor-tutorial-**barometer-sensor**-2.html
* https://androidkennel.org/android-sensors-game-tutorial/

**Sensors**

* http://www.dropsens.com/en/home.html?gclid=COvrp7G42tMCFc4WaAodSNUDCQ
* http://www.vericomcomputers.com/home/?gclid=CMyonbG42tMCFUsfaAodmQIB\_w
* http://www.appsrox.com/android/tutorials**/accelerometer**-golf/
* https://code.tutsplus.com/tutorials/using-the**-accelerometer-on-android--mobile**-22125
* http://www.vogella.com/tutorials/AndroidSensor/article.html
* http://www.techotopia.com/index.php/A\_Basic\_Overview\_of\_Android\_Threads\_and\_Thread\_handlers
* http://androidexample.com/Thread\_With\_Handlers\_-\_Android\_Example/index.php?view=article\_discription&aid=58
* https://www.101apps.co.za/index.php/articles/using-threads-tutorial.html
* http://stackoverflow.com/questions/4722974/threading-example-in-android
* http://stackoverflow.com/questions/1921514/how-to-run-a-runnable-thread-in-android
* https://examples.javacodegeeks.com/android/core/thread/android-thread-example/
* http://www.vogella.com/tutorials/A**ndroidBackgroundProcessi**ng/article.html

**Chapter Seven Networking, Web Service, Remote APIs**

Objective How to share between 2 or more devices with Wi-FI Direct

Background

Android lets your application connect to the internet or any other local network and allows you to perform network operations. A device can have various types of network connections. This chapter focuses on using either a Wi-Fi or a mobile network connection.

## Checking Network Connection

Before you perform any network operations, you must first check that are you connected to that network or internet e.t.c. For this android provides **ConnectivityManager** class. You need to instantiate an object of this class by calling **getSystemService()** method. After checking that you are connected to the internet, you can perform any network operation.

[**Fast Android Networking Library**](https://github.com/amitshekhariitbhu/Fast-Android-Networking)**?**

* Recent removal of HttpClient in Android Marshmallow(Android M) made other networking library obsolete.
* No other single library do each and everything like making request, downloading any type of file, uploading file, loading image from network in ImageView, etc. There are libraries but they are outdated.
* As it uses [OkHttp](http://square.github.io/okhttp/), most important it supports **HTTP/2**.
* No other library provides simple interface for doing all types of things in networking like setting priority, cancelling, etc.

Fast Android Networking Library is a powerful library for doing any type of networking in Android applications which is made on top of **OkHttp Networking Layer**.

## Advantages of using Fast Android Networking library over other libraries:

* **OkHttpClient** can be customized for every request easily — like timeout customization, etc. for each request.
* As it uses **OkHttpClient** and **Okio**, it is faster.
* A single library for all types of networking — download, upload, multipart.
* Supports JSON Parsing to Java Objects (also support Jackson Parser).
* *Proper Response Cachi*ng — which leads to reduced bandwidth usage.
* Complete analytics of any request can be obtained. You can know bytes sent, bytes received, and the time taken on any request. These analytics are important so that you can find the data usage of your application and the time taken for each request, so you can identify slow requests.
* You can get the current bandwidth and connection quality to write better logical code — download high quality images on excellent connection quality and low on poor connection quality.
* An executor can be passed to any request to get a response in another thread. If you are doing a heavy task with the response, you cannot do that in main thread.
* **Prefetching** of any request can be done so that it gives instant data when required from cache.
* All types of customization are possible.
* Proper request cancelling.
* A ***simple interface*** to make any type of request.
* Supports **RxJava**.

## Why should you use Fast Android Networking Library?

* Recent removal of HttpClient in Android Marshmallow(Android M) made other networking library obsolete.
* No other single library do each and everything like making request, downloading any type of file, uploading file, loading image from network in ImageView, etc. There are libraries but they are outdated.
* As it uses OkHttp , most important it supports HTTP/2.
* No other library provides simple interface for doing all types of things in networking like setting priority, cancelling, etc.

**Chapter Seven Networking, Web Service, Remote APIs**

**Web services are open standard** (XML, SOAP, HTTP etc.) based Web applications that interact with other web applications for the purpose of exchanging data. Web Services can convert your existing applications into Web-applications. Creating **web service application** in android is not a difficult task. We can easily create a restful web service application in android to authenticate or save information into the external database such as oracle, mysql, postgre sql, sql server using other application developed in java, .net, php etc languages.

A web service is a **standard for exchanging** information between different types of applications irrespective of language and platform. For example, an android application can interact with java or .net application using web services. A web service is a standard used for exchanging information between applications or systems of heterogeneous type. Software applications written in various programming languages and running on various platforms can use web services to exchange information over Internet using http protocol. This inter-operability can be achieved between Java and Dot net applications, or PHP and Java applications.

Here are the reasons for using Web Service:

* **Expose method as a service over network:**

Web service is a chunk of code written in some programming language (Say C# or Java) that can be invoked remotely through http protocol. Once the Web methods are exposed publically, any applications can invoke it and use the functionality of the web methods.

* **Application Inter-operability – Connect heterogeneous applications:**

With the help of web service, heterogeneous applications (Java and Dot Net / Java and PHP application) can communicate with each other. Web service written in Java can be invoked from PHP by passing the required parameters to web methods. This makes the applications platform and technology independant.

* **Standardized protocol:**

Web Services uses standardized industry standard protocol for the communication which must be adhered and followed by the applications creating Web Service.

* **Cost effective communication:**

Web service is using SOAP over HTTP protocol for communication which is much cheaper than systems like B2B.

**Existing Web applications are in a need of creating mobile applications** to show their presence in mobile platform as well. Almost all web applications are having their Mobile applications created in Android, iOS or Windows platform. Exposing the existing functionalities of the applications is bit tough as all the functionalities have to re-written in the respective platforms. But it can be easily achieved with much ease by creating Web Service and expose the existing functionalities as web methods to Mobile platforms.

Here are **the few advantages of using Web Service in Android**:

* **Make client more lightweight:**

Adding a web service layer makes the client more lightweight, both in terms of the required CPU power and the bandwidth used during the processing. Most of the processing to be done in client end can be separated and put inside a web service layer which will be extremely helpful for end-users in terms of:

* + - Using less CPU increases the battery life
    - Using less bandwidth reduces monthly payments over data charge
* **Re-usage of existing functionalities:**

While designing the web service, we could also get significant benefits by reusing the existing functionalities by exposing them as web methods.

* **Remote DB hit made simple:**

DB residing remotely can be hit from inside Android applications through Web Service calls.

Making Web Service call from Android applications allows us to add functionality outside the scope of a DB like caching data, applying business rules over the data etc.,

**Chapter Seven Networking, Web Service, Remote APIs**

**How to Invoke Web Services from Android**

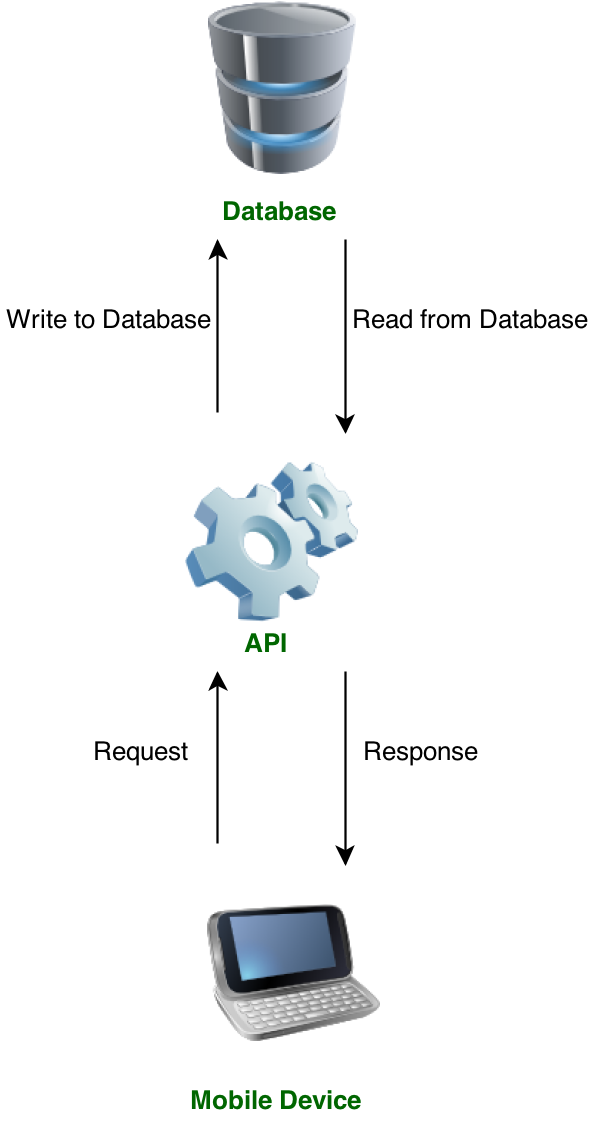
There are two ways in which invoke web services

* **Raw APIs**  : Use the HttpClient and XML parser to manually create a soap request and parse the soap response.
* Using a soap client library : like **KSOAP library** which does the low level work for parsing and dealing with soap messages – For Android, there is library available at <http://code.google.com/p/ksoap2-android/> . Its good to see some active development for KSOAP 2, I remembered I wrote the first article on KSOAP 2 way back in 2003
* ( <http://naveenbalani.com/index.php/2010/05/deliver-web-services-to-mobiles/>)and good to see it back in development for android.

To build an Android client that consumes a Spring-based RESTful web service. Specifically, the client will consume the service created in Building a **RESTful Web Service**. **RemoteControlClie**nt enables exposing information meant to be consumed by remote controls capable of displaying metadata, artwork and media transport control buttons.

The Java SDK includes a library called Remote API that lets you transparently access App Engine services from any Java application. For example, you can use Remote API to access a production datastore from an app running on your local machine. You can also use Remote API to access the datastore of one App Engine app from a different App Engine app.

Diagram on Remote API



**Chapter Seven Networking, Web Service, Remote APIs**

Topics to learn

* + React to network state, check for connectivity and receive changes in connectivity
  + ***HttpUrlConnection,*** Volley, OkHttp, SPDY, Web Sockets
  + Power Efficient Network Polling
  + Server-side Push
  + Android Wi-Fi - Network Service Discovery, Wi-Fi Direct,
  + Web Content - customize web browser, http get and post, use WebView and parse JSON and XML
  + Google Static Maps v2
  + Social Networking read owner profile, integrate twitter, Facebook, FB SDK for Android
  + FourSquare API using OAuth2
  + Find Online Web Services and API

Illustrated Example

Networking and Fast Networking

* https://www.raywenderlich.com/126770/android-networking-tutorial-getting-started
* https://blog.mindorks.com/simple-and-fast-android-networking-19ed860d1455
* https://amitshekhariitbhu.github.io/Fast-Android-Networking/
* http://stackoverflow.com/questions/16902716/comparison-of-android-networking-libraries-okhttp-retrofit-and-volley
* http://www.vogella.com/tutorials/AndroidNetworking/article.html

Web Service

* https://www.javatpoint.com/android-web-service
* http://programmerguru.com/android-tutorial/android-webservice-tutorial/
* https://www.codeproject.com/Articles/112381/Step-by-Step-Method-to-Access-Webservice-from-Android
* https://dzone.com/articles/invoke-webservices-android
* https://spring.io/guides/gs/consuming-rest-android/
* http://stackoverflow.com/questions/10588311/android-web-service-tutorial
* http://www.androidhive.info/2012/01/android-login-and-registration-with-php-mysql-and-sqlite/
* http://androidexample.com/Restful\_Webservice\_Call\_And\_Get\_And\_Parse\_JSON\_Data-\_Android\_Example/index.php?view=article\_discription&aid=101
* https://www.tutorialspoint.com/webservices/

Android Remote APIs

* https://code.tutsplus.com/tutorials/android-sdk-making-remote-api-calls--mobile-17568
* https://cloud.google.com/appengine/docs/standard/java/tools/remoteapi
* https://www.chrisblunt.com/android-consuming-a-remote-json-api-with-volley/

**Chapter Eight - Part One Maps, Location and Activity APIs**

Objective Learn about how to incorporate Google Maps and Geo Fencing and other APIs.

Background

The Google Location Services API, part of Google Play Services, provides a more powerful, high-level framework that automatically handles location providers, user movement, and location accuracy. It also handles location update scheduling based on power consumption parameters you provide. In most cases, you'll get better battery performance, as well as more appropriate accuracy, by using the Location Services API.

Obtaining user **location from a mobile device can be complicated**. There are several reasons why a location reading (regardless of the source) can contain errors and be inaccurate. Some sources of error in the user location include:

* **Multitude of location sources**

GPS, Cell-ID, and Wi-Fi can each provide a clue to users location. Determining which to use and trust is a matter of trade-offs in accuracy, speed, and battery-efficiency.

* **User movement**

Because the user location changes, you must account for movement by re-estimating user location every so often.

* **Varying accuracy**

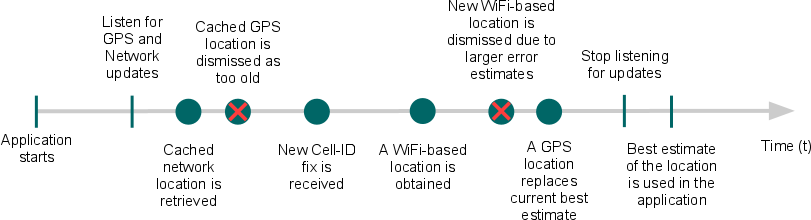
Location estimates coming from each location source are not consistent in their accuracy. A location obtained 10 seconds ago from one source might be more accurate than the newest location from another or same source.

These problems can make it difficult to obtain a reliable user location reading. This document provides information to help you meet these challenges to obtain a reliable location reading. It also provides ideas that you can use in your application to provide the user with an accurate and responsive geo-location experience

Here's the typical flow of procedures for obtaining the user location:

1. Start application.
2. Sometime later, start listening for updates from desired location providers.
3. Maintain a "current best estimate" of location by filtering out new, but less accurate fixes.
4. Stop listening for location updates.
5. Take advantage of the last best location estimate.

Figure below demonstrates **this model in a timeline that visualizes the period** in which an application is listening for location updates and the events that occur during that time.



**Chapter Eight - Part One Maps, Location and Activity APIs**

Android gives your applications access to the location services supported by the device through classes in the ***android.location*** package. The central component of the location framework is the **LocationManager** system service, which provides APIs to determine location and bearing of the underlying device.

With the **Google Maps Android API**, you can add maps to your app that are based on Google Maps data. The API automatically handles access to Google Maps servers, data downloading, map display, and touch gestures on the map. You can also use API calls to add markers, polygons and overlays, and to change the user's view of a particular map area. To integrate Google Maps into your app, you need to install the Google Play services libraries for your Android SDK.

**A View which displays a map** (with data obtained from the Google Maps service). When focused, it will capture **keypresses and touch gestures** to move the map. Users of this class must forward all the life cycle methods from the Activity or Fragment containing this view to the corresponding ones in this class. Android allows us to integrate Google maps in our application. You can show any location on the map , or can show different routes on the map etc. You can also customize the map according to your choices.

Google made new Maps V2 API as a part of **Google Play Services** SDK. So before we start developing maps we need to download google play services from SDK manger. You can open SDK manager either from Eclipse or from android sdk folder. Google provides via Google play a library for using Google Maps in your application. The following description is based on the Google Maps Android API v2 which provides significant improvements to the older API version. The library provides the ***com.google.android.gms.maps.MapFragment*** class and the MapView class for displaying the map component. You need to add additional information to your AndroidManifest.xml file to use Google Maps.

**---------------------------------------------------------------------------------------------------------------------------------------**

**Geofencing** combines awareness of the user's current location with awareness of the user's proximity to locations that may be of interest. To mark a location of interest, you specify its latitude and longitude. To adjust the proximity for the location, you add a radius. The **latitude, longitude, and radius define a geofence,** creating a circular area, or fence, around the location of interest. Location aware resources allow your application to interact with the physical world and they are ideal for increasing user engagement. Although many mobile apps use them, the topic of this tutorial is a feature that is often overlooked, [geofencing](https://en.wikipedia.org/wiki/Geo-fence). A **geofence** is a virtual perimeter set on a real geographic area. Combining a user position with a geofence perimeter, it is possible to know if the user is inside or outside the geofence or even if he is exiting or entering the area.

How to use geofences on Android by creating an application that shows the user a notification when they enter or exit a geofence. On Android, there are several ways to work with geofences. You could even create your own implementation to work with geofences, but it is easier to use Google's Geofencing API.

**Geofence** is an interface that represents a geographical area that should be monitored. It is created by using the Geofence.Builder. During its creation, you set the monitored region, the geofence's expiration date, responsiveness, an identifier, and the kind of transitions that it should be looking for.

**Geo-fencing** is a feature in a software program that uses the global positioning system (GPS) or radio frequency identification (RFID) to define geographical boundaries. A geofence is a virtual barrier. Represents a geographical region, also known as a geofence. Geofences can be monitored by **geofencer** service. And when the user crosses the boundary of a geofence, an alert will be generated.

**Chapter Eight - Part One Maps, Location and Activity APIs**

Topics to learn

Location, retrieve last location, update location upon change, list all enabled providers, translate a location to an address (reverse geo-coding) and translate address to location (geocoding)

Access GPS information and mocking GPS coordinates

Use Google Maps - add maps to app, add markets, view, set up proximity alert

Add multiple location markets on Google MapView

Create overlays and change modes for Google MapView, implement location search on Google Maps

Use OpenStreetMap, create overlays, use a scale, handle touch events, get location updates, etc.

Fused Location Manager

Google Maps v2, Work with markers, draw circles, polygons, useful location API utilities, Geocoding

Use LocationClient

Geo Fencing

Activity Recognition

Illustrated Example

* https://developer.android.com/guide/topics/location/index.html
* http://www.cs.dartmouth.edu/~campbell/cs65/lecture17/lecture17.html
* http://www.vogella.com/tutorials/AndroidLocationAPI/article.html
* https://dzone.com/articles/android-location-based
* https://examples.javacodegeeks.com/android/core/location/android-location-based-services-example/
* https://www.tutorialspoint.com/android/android\_location\_based\_services.htm

Geo Fencing

* https://developer.android.com/training/location/geofencing.html
* https://code.tutsplus.com/tutorials/how-to-work-with-geofences-on-android--cms-26639
* https://dzone.com/articles/android-geofencing-google-maps
* https://developers.google.com/android/reference/com/google/android/gms/location/Geofence
* https://software.intel.com/en-us/android/articles/implementing-map-and-geofence-features-in-android-business-apps
* http://stackoverflow.com/questions/26507424/very-easy-android-geofence-example

Google Maps

* https://developers.google.com/android/reference/com/google/android/gms/maps/MapView
* http://stackoverflow.com/questions/16536414/how-to-use-mapview-in-android-using-google-map-v2
* https://www.tutorialspoint.com/android/android\_gestures.htm
* http://www.vogella.com/tutorials/AndroidGoogleMaps/article.html
* http://www.androidhive.info/2013/08/android-working-with-google-maps-v2/
* https://inducesmile.com/android/android-mapview-example-tutorial/
* http://www.vogella.com/tutorials/AndroidGoogleMaps/article.html

**Chapter Eight - Part Two String Localization and Internationalization**

Objective To learn how to manage localization of string in layout, dates, times, etc.

Background

Android will run on many devices in many regions. To reach the most users, your application should handle text, audio files, numbers, currency, and graphics in ways appropriate to the locales where your application will be used.

Apps include resources that can be specific to a particular culture. For example, an app can include culture-specific strings that are translated to the language of the current locale. It's a good practice to keep culture-specific resources separated from the rest of your app. Android resolves language- and culture-specific resources based on the system locale setting. You can provide support for different locales by using the resources directory in your Android project.

An android application can run on many devices in many different regions. In order to make your application more interactive, your application should handle text, numbers, files, etc. in ways appropriate to the locales where your application will be used.

Localization is more than just the translation of the strings in your application in other languages. It also involves displaying data, such as dates and times, in the right format for your users. Even if you only want an English version of your application it's good to know what localization involves. That way you'll be ready if you decide to add support for additional locales and you won't have to look for strings and values all over your application's code base.

Topics to learn

* [Application localization process](https://www.icanlocalize.com/site/tutorials/android-application-localization-tutorial/#1)
* Localizing strings in layouts
* [Translating strings](https://www.icanlocalize.com/site/tutorials/android-application-localization-tutorial/#2)
* Localizing dates and times
* [Localizing images](https://www.icanlocalize.com/site/tutorials/android-application-localization-tutorial/#3)
* [Running and testing the localized application](https://www.icanlocalize.com/site/tutorials/android-application-localization-tutorial/#4)
* [Localization checklist](https://www.icanlocalize.com/site/tutorials/android-application-localization-tutorial/#5)
* [Sample multilingual Android app for download](https://www.icanlocalize.com/site/tutorials/android-application-localization-tutorial/#sample-app)

Illustrated Example

* http://navinsandroidtutorial.blogspot.in/2014/04/android-localization-tutorial.html
* http://www.ibabbleon.com/android\_app\_localization\_translation.html
* https://code.tutsplus.com/tutorials/how-to-localize-an-android-application--cms-22154
* http://programmerguru.com/android-tutorial/android-localization-example/
* http://www.androidhive.info/2014/07/android-building-multi-language-supported-app/
* https://code.tutsplus.com/tutorials/how-to-localize-an-android-application--cms-22154

Reference

* https://developer.android.com/guide/topics/resources/localization.html
* https://developer.android.com/training/basics/supporting-devices/languages.html
* https://www.tutorialspoint.com/android/android\_localization.htm
* https://www.icanlocalize.com/site/tutorials/android-application-localization-tutorial/

**Chapter Nine Data Storage Methods**

Objective To store data on local device

Background

Data persistence refers to storing data and serialization refers to how data is represented in its stored state. There are two common techniques for persistence and serialization in Android these are SQLite and Shared Preferences. There also two methods that allow you to transport data over a network or between two devices. The files backing your **SharedPreferences** objects are regular XML files stored in app's data directory. The structure is simple because it allows only key/ value pairs to be stored. Android APIs also provide a convenient abstraction that allows you to read and write data in a type-safe way.

In many apps it is common to provide a separate UI that allows users to change the options and settings of the app using classes like PreferenceActivity and PreferenceFragment. Example - building an Android Settings Screen - https://medium.com/@JakobUlbrich/building-a-settings-screen-for-android-part-1-5959aa49337c or https://examples.javacodegeeks.com/android/core/ui/settings/android-settings-example/

To make a **SQLite database** to store data it is smart to create a Content Provider so as to use several Android utility and UI-related classes for easier code writing. When an Android database design follows the first three normal forms it is considered ***normalized***.

If you wrap your SQLite databases using Content Provider component you can manage all calls to the database from one place and use several readymade utilities for working with databases example the Database class is extends SQLiteOpenHelper class includes two methods onCreate() and onUpgrade(). Use ContentResolver.query() method it calls ContentProvider.query() method. When you execute a piece of SQL on a SQLite database you perform a ***transaction***, remember most calls to a ContentProvider result in only one SQL statement there is no need to manage your own transactions manually and will execute multiple SQL statements there you need to manage your own transactions.

***Binary data*** represents anything that cannot be represented using simple data types in Java usually an image or some other media file. Storing Binary data in ContentProvider I refer to as using the Media Store.

Then there is **another** challenge - to send data over the Internet or share it with another device - then firstly the data is re-formatted by a technique called **serialization** to allow the receiving end to work with and suitable for send over the Internet. Android provides its internal serialization API with the *Parcelable* interface which is not good for persistent storage or for transferring over a network. Instead you could use two suitable formats for persisting complex data - JSON and Google Protocol Buffers - both are well supported by Android. **JSON** is excellent for representing complex objects (not binary values) and is the de-facto standard for sharing data with web services. A JSON array with 3 objects is stored in a class and the data can be read using an *InputStream* with the **JsonReader** API. Alternatively the entire contents of *InputStream* can be read and put to a String then pass that to a constructor of **JSONArray**. Likewise **JsonWriter** class allows you to write JSON data to *OutputStream*. The limitations of JSONObject and JSONArray classes are overcome using an open-source library you can handle advanced serialization and deserialization of JSON data - this is **Gson**. A typical solution is to use a library to convert a POJOs - Plain Old Java Objects - into JSON and back - here your define your data as regular Java Objects with getters and setters and include the **Gson** library.(p182- pushing limits).

**Google Protocol Buffer** or PROTOBUF is a way of encoding structured data in an efficient and extensible format. It supports binary data mixed with simple data types and also has an advanced and extensible schema support. **Protobuf** has implementation including a lite Java variant for Android. Example http://www.java2s.com/Open-Source/Android\_Free\_Code/Example/example/jgilfelt\_android\_protobuf\_example.htm and https://developers.google.com/protocol-buffers/docs/tutorials and https://github.com/jgilfelt/android-protobuf-example.

**Chapter Nine Data Storage Methods**

After you resolve all your persistence needs for your app consider if the app needs to use backup services for Android provided by Google - to back up persistence data and restore it in case of a factory reset or upgrades to a new Android device. Android provides API for managing backups and the backup is stored in cloud and can be restored only to a device with the same Google ID. This is application data backup.

Topics to learn

* + Shared Preferences
  + SQLite Database
    - Create, Insert Values, Loading Values from another source
    - Work with dates in SQLite
    - Parse JSON using JSONObject
    - Parse XML using DOM API, XmlPullParser
    - Add a contact and read contact data
  + Content Provider
    - File Saving and Loading
    - Data Persistence
    - Get File info, read a file shipped with app
    - List a directory
    - Create and using content provider
    - Using Media Store provider
    - Using Contacts provider

Illustrated Example

* https://www.tutorialspoint.com/android/android\_shared\_preferences.htm
* http://www.journaldev.com/9412/android-shared-preferences-example-tutorial
* http://androidexample.com/Android\_SharedPreferences\_Basics/index.php?view=article\_discription&aid=126
* http://androidopentutorials.com/android-sharedpreferences-tutorial-and-example/

**Chapter Ten In-App Billing**

Objective

Background

**Complete these steps to implement In-app Billing in your application:**

* **Add** the In-app Billing library to your project.
* Update your AndroidManifest.xml file.
* Create a ServiceConnection and bind it to the **IInAppBillingService** .
* Send In-app Billing requests from your application to IInAppBillingService

Creating app that have ability to up-sell, allow the purchase of items or add functionality can fill an important role in apps marketing strategy. Google has provided an API that developers can use to add in-app billing to apps.

Topics to learn

* + Google Play In-App Billing - install Google's service
  + Add In-App billing to an Activity
  + Listing items for In-App purchase and Monetize your app

Illustrated Example

* https://developer.android.com/google/play/billing/billing\_integrate.html
* http://stackoverflow.com/questions/8735931/how-to-implement-in-app-billing-in-an-android-application
* https://blahti.wordpress.com/2014/07/30/how-to-add-in-app-billing-in-android-part-1/
* http://www.techotopia.com/index.php/Integrating\_Google\_Play\_In-app\_Billing\_into\_an\_Android\_Application\_%E2%80%93\_A\_Tutorial
* http://www.techotopia.com/index.php/An\_Android\_Studio\_Google\_Play\_In-app\_Billing\_Tutorial
* http://www.theappguruz.com/blog/implement-in-app-purchase-version-3
* https://hackernoon.com/implementing-in-app-billing-in-android-4896232c7d6b

**Chapter Eleven Push Messages**

Objective - To understand the best push messages

A notification is a message you can display to the user outside of your application's normal UI. You can create your own notifications in android very easily. Android provides **NotificationManager** class for this purpose. In order to use this class, you need to instantiate an object of this class by requesting the android system through **getSystemService() method**. Its syntax is given below −

NotificationManager NM;

NM=(NotificationManager)getSystemService(Context.NOTIFICATION\_SERVICE);

After that you will create Notification through **Notification** class and specify its attributes such as icon, title and time etc. Its syntax is given below −

Notification notify = new Notification(android.R.drawable.stat\_notify\_more,title,System.currentTimeMillis());

The next thing you need to do is to create a **PendingIntent** by passing context and intent as a parameter. By giving a PendingIntent to another application, you are granting it the right to perform the operation you have specified as if the other application was yourself.

PendingIntent pending = PendingIntent.getActivity(getApplicationContext(), 0, new Intent(),0);

The last thing you need to do is to call **setLatestEventInfo** method of the Notification class and pass the pending intent along with notification subject and body details. Its syntax is given below. And then finally call the notify method of the NotificationManager class.

notify.setLatestEventInfo(getApplicationContext(), subject, body,pending);

NM.notify(0, notify);

Topics to learn

Google Cloud Messaging Setup - Prepare for GCM

Send and Receive Push Messages - prepare Manifest file

Receive Messages - Add Broadcast Receiver class, Add IntentService class and register a device

Send Messages - send text message with and without AsycTask

Use of Node Server and a Push Message Service example

Illustrated Example

* https://code.tutsplus.com/tutorials/how-to-get-started-with-push-notifications-on-android--cms-25870
* http://www.androidhive.info/2012/10/android-push-notifications-using-google-cloud-messaging-gcm-php-and-mysql/
* http://api.shephertz.com/tutorial/Push-Notification-Android/
* https://www.intertech.com/Blog/push-notifications-tutorial-for-android-using-google-cloud-messaging-gcm/
* https://quickblox.com/developers/SimpleSample-messages\_users-android
* https://docs.kii.com/en/samples/push-notifications/push-notifications-android/

References

* https://www.tutorialspoint.com/android/android\_push\_notification.htm
* https://code.tutsplus.com/tutorials/how-to-get-started-with-push-notifications-on-android--cms-25870
* https://developers.google.com/cloud-messaging/

**Chapter Twelve Communicating with Remote Devices**

Objective - To communicate between Android devices with NFC and Bluetooth, to communicate with Android device from another device like laptop with USB and to use Wi-Fi Direct.

Background

This is about my Android device and the other Android device ( another smartphone, a wearable, a glass, a tv with Android, another laptop and so on). To communicate between a laptop and my Android device I can use a USB cable. I can access Wi-Fi on my Android device at any place like an airport, a metro station, at home, etc. I can share data and much more with my friends phone using Bluetooth.

You can use APIs in Android to perform raw serial communication over USB or Android Open Accessory Protocol as in Accessory Development Kit (ADK). https://source.android.com/devices/accessories/protocol

Android devices support Classic Bluetooth profiles good for audio streaming. There is support of Bluetooth Low Energy devices and Bluetooth Smart Technology which allows you to write apps that can discover and communicate with devices that support the GATT profile like heart monitors, step counters, other low-power accessories.

Android has **three modes** in which its Wi-Fi can operate: **infrastructure** (standard Wi-Fi connected to an access point), **tethering** (Android device acting as a Wi-Fi access point to other devices) and **Wi-Fi Direct** allows an app to set up a peer-to-peer network that works without the need for a dedicated access point which makes network communication between devices in a more ad hoc scenario. Useful in cases like a multi-player communication or if you need a secure and safe way of sharing data (like photos) between two friends.

**Wi-Fi Direct**

* https://trivedihardik.wordpress.com/2012/08/31/android-wifi-direct-example/
* https://github.com/mayfourth/WiFi-Direct-File-Transfer
* https://github.com/anuj7sharma/WiFiDIrectDemo
* http://stackoverflow.com/questions/17715283/transfer-data-between-android-devices-using-wifi-direct

Topics to learn

* + - On-Device Web Services
      * RESTful with Restlet
      * Web Socket Server
    - Use of NodeJS, Express as Server and SOA
    - Wi-Fi Direct

**Chapter Thirteen How to Write a Test Case and Use it in Android**

Objective Learn a lot of things about testing in Android and become a great tester

Background

There are many types of tests like Unit Testing that tests individual components in isolation. JUnit and TestNG are frameworks used here. Mock Objects are used where interaction with other components is required. Java has several popular mocking frameworks. Terms like NPE (is the Java acronym for Null Pointer Exception) and ANR (is Android acronym for App Not Responding) and FC - force close - when Android requires you close a failed app.

TDD - Test Driven Development in Android

Issue - right now mocking frameworks in vogue include Robotium and EasyMock, could there be a better framework.

Solution - one approach is in-direct method wherein you set up two test projects - one test project uses Android Studio for UI-related tests and the other test project for mock supported tests. Then extract as much logic as possible to the classes that can be unit-tested. Alternatively to conduct mocking tests is to use a local server like IIS from Microsoft, Node server or LAMP stack and conduct tests which are more like integration testing then unit testing. A few suggestions on how to conduct a mocking test in Android could be found in the links below:

* Get started with testing -
  + Two test types - local unit tests and instrumented tests (run on device or emulator) and
  + Test APIs for JUnit, Android Testing Support Library (AndroidJUnitRunner, Espresso, UI Automator) and
  + Assertion Classes and
  + Monkey and monkeyrunner for functional level testing in Android and
  + Guides to building the different tests

https://developer.android.com/training/testing/start/index.html

* Use the API class *android*.*test.mock* - https://developer.android.com/reference/android/test/mock/package-summary.html
* Build local tests by setting up your testing environment using ( In your Android Studio project, you must store the source files for local unit tests at **module-name**/src/test/java/. This directory already exists when you create a new project.) and create a local unit test class (To create a basic JUnit 4 test class, create a Java class that contains one or more test methods. A test method begins with the @Test annotation and contains the code to exercise and verify a single functionality in the component that you want to test) and to make sure you test only your code and do not depend on any particular behavior of the Android platform use a mocking framework like Mockito v1.9.5 include a mocking object and run local unit tests. https://developer.android.com/training/testing/unit-testing/local-unit-tests.html
* Test your app - https://developer.android.com/studio/test/index.html
* Building effective unit tests - https://developer.android.com/training/testing/unit-testing/index.html
* A tutorial - http://www.vogella.com/tutorials/AndroidTesting/article.html
* http://www.vogella.com/tutorials/Mockito/article.html
* https://www.toptal.com/android/testing-like-a-true-green-droid
* Android arsenal - https://android-arsenal.com/tag/67

Topics to learn

**xliii.** Android Testing Principles - What to test - Basic Unit Testing

**xliv.** Testing Activities, Test Services, Test Content Providers

**xlv.** Running Tests

**xlvi.** Continuous Integration and how to such a CI system

**xlvii.** Android Testing Guidelines

**Chapter Fourteen Google Play Services and Distributing the App**

Objective

[1] To learn how to get the authorization to the online services such as Google Drive, Google Cloud Platform, Google Play Services API and how to use Google Cloud Messaging service to send a push notifications AND [2] How to more efficiently distribute your app

With **Google Play services**, your app can take advantage of the latest, Google-powered features such as Maps, Google+, and more, with automatic platform updates distributed as an APK through the Google Play store. This makes it faster for your users to receive updates and easier for you to integrate the newest that Google has to offer. Google Play services **gives you the freedom to use the newest APIs for popular Google services** without worrying about device support. Updates to Google Play services are distributed automatically by the Google Play Store and new versions of the client library are delivered through the Android SDK Manager. This makes it easy for you to focus on what's important: your users' experience.

**Add** Google Play Services APIs in your project by adding the following to your Gradle build file dependencies section *compile 'com.google.android.gms:play-sercies:3.1.36'*

***Authorization*** *-* first step : Retrieve an authorization token for the Google account of the user, to do so you first need to fetch the account name for the user. To see code sample see source code for this chapter.

***Google Drive App Data*** - If you have a Google account you can access Google Drive cloud storage service. The first step is to add your app's package name and SHA1 key string in the Google API Console. Then create a new Client ID and add information needed. Then add the necessary dependencies to your Gradle build file *compile 'com.google.api-client:google-api-client-android:1.16.0-rc' and compile 'com.google.apis:google-api-services-drive:v2-rev89-1.15.0-rc '.*

***Google Cloud Endpoints*** *-* Using this service it is easy to get started with building a cloud backend for your mobile app. Step 1 - you generate a basic App engine backend from Android Studio. After code generation is complete the result is two modules - "endpoint" and "AppEngine" then add new POJOs with getters and setters - this is your TaskInfo class to AppEngine module in project then run the "Generate Endpoint" followed by "Generate Client Libraries" from Tools Menu again. The result is the client-side libraries to use in your app.

***Google Cloud Messaging*** - To integrate GCM in your app enable your Google API Console and set up a Server key for your app under API access. Make sure that Google Cloud Messaging is enabled in your project.

Google provides a number of useful tools, guides, and checklists that will help you in the process of getting your app out to your users. To gain access to the Google Play Console you must first register for a publisher account and set a Google Wallet Merchant Account if you want to sell apps or support in-app purchases.

Monetizing your app happens in several ways -

[1] Sell your app for a price or

[2] Make your app FREE and allow users to buy additional content from within your app - IN-APP Billing

[3] Bring ads to your app using AdMob services first sign up to www.google.com/ads/admob and then download the AdMob SDK for Android and add the appropriate dependency to your Gradle build script. Next add the permissions as well as AdActivity to your manifest and use com.google.ads.AdView class and use the same class in XML layouts - use AdView in XML layout

Then target the ads by giving AdRequest a number of parameters like making sure all ads are child safe. It is possible to change color of ads, and use interstitial ads.

Perform **App Licensing check** to avoid the obvious pitfalls

**APK Expansion files** - go over the 50MB limit of APK files go up to 4GB of additional data. Learn to create and download expansion files.

Topics to learn

* Ads in Android Apps - Target Ads, Ad Colors, Interstitial Ads and Application Licensing
* APK Expansion files - create and download expansion files

References

* https://code.tutsplus.com/articles/android-from-scratch-google-play-services--cms-26040
* https://www.lynda.com/Google-Play-Services-tutorials/Google-Play-Services-Android/474086-2.html
* https://developer.android.com/distribute/google-play/work.html
* https://developer.android.com/google/play/dist.htmlhttps://developers.google.com/android/guides/overview

**Chapter Fifteen Using Dependencies like Picasso, Butterknife, Cupboard**

INDISPENSABLE ANDROID LIBRARIES

1. Dagger
2. Butterknife
3. Cupboard
4. Parceler
5. Picasso
6. Tape
7. Android Support Libraries

Check out - http://blog.patrickbaumann.com/2014/03/indispensable-android-libraries/

**xlxi**. How to add Picasso Library in Android Studio

* http://stackoverflow.com/questions/28603191/how-to-add-picasso-library-in-android-studio
* https://www.simplifiedcoding.net/picasso-android-tutorial-picasso-image-loader-library/
* https://teamtreehouse.com/community/adding-picasso-in-android-studio
* https://code.tutsplus.com/tutorials/android-sdk-working-with-picasso--cms-22149

**xlxii.** Use Butterknife for View Injection in Android

* http://www.vogella.com/tutorials/AndroidButterknife/article.html
* https://www.sitepoint.com/tidying-code-with-android-butterknife/
* http://www.journaldev.com/10439/android-butter-knife-example-tutorial
* http://www.thekeyconsultant.com/2013/09/5-reasons-you-should-use-butterknife.html
* https://code.tutsplus.com/tutorials/quick-tip-using-butter-knife-to-inject-views-on-android--cms-23542

**xlxiii.** Cupboard: Simple Persistence for Android

* http://code.neenbedankt.com/introducing-cupboard-simple-persistance-for-android/
* https://www.objc.io/issues/11-android/sqlite-database-support-in-android/
* https://guides.codepath.com/android/Easier-SQL-with-Cupboard

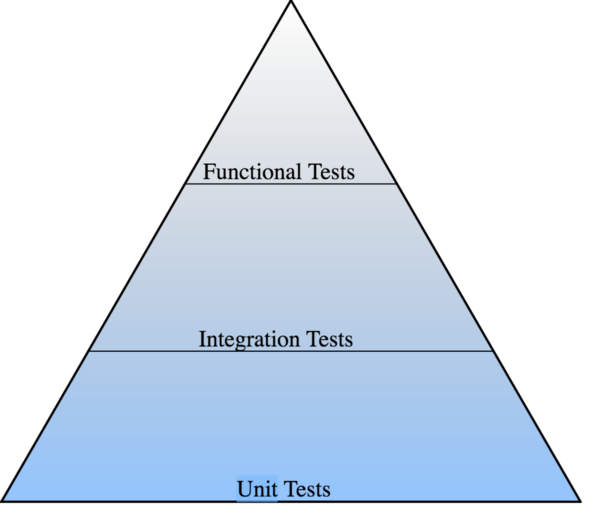
**More on Android Testing**

**Test-Driven Development** takes a different approach to software development and its principles lead to high quality software. With TDD you start by defining your tests based on how your app will be used. TDD involves a number of tools.

Unit and Automated Testing in Android

**Automation tests** are important as it ensures quality while developing application.TDD enforces writing tests before writing implementation. Refer this [previous tutorial](https://medium.com/@nileshjarad/why-developers-scared-to-refactor-code-47efd1b854e7#.m4ot7zaaq) for more information about **TDD**.

**Testing pyramid :** As we can see Pyramid below,



***Unit tests***cover large section of pyramid making solid base. Unit tests are easy to write.

***Integration Testing***ensures integration of modules is correct and cover second large piece of Pyramid.

***Functional testing***usually describes *what* the system does and occupies smallest amongst the three part.

There are many testing tools and frameworks that can be used while developing application. Some of them are provided & supported by Google and some of them are third party .

***Types of testing in Android :***

**Unit Test**

Unit tests mainly target smallest functionality (like method, class , component or small module) with isolation from other component.

*Tools used for Unit Test*

* [JUnit](http://junit.org/junit4/) is a simple framework to write repeatable tests. It is an instance of the JUnit architecture for unit testing frameworks.
* [Robolectric](http://robolectric.org/) is popular Android unit test framework that allows faster test execution by running tests on the JVM (no device or emulator needed).

**Instrumentation Test**

Instrumentation Test (UI) are tests that mock User Interaction like clicking button , Typing text in EditText. Android Instrumentation is a set of “hooks” into the Android system that allows you to control the lifecycle of Android components (i.e. drive the activity lifecycle yourself instead of having these driven by the system). These tests require an actual device or emulator to run .

*Tools used for Instrumentation Test*

* [Espresso](https://github.com/codepath/android_guides/wiki/UI-Testing-with-Espresso) — Android UI Test Framework provided by Google that handles test synchronization very well.
* [UIAutomator](https://developer.android.com/training/testing/ui-testing/uiautomator-testing.html) — Android UI Test Framework provided by Google for testing across multiple apps at the same time.
* [Robotium](https://github.com/codepath/android_guides/wiki/UI-Testing-with-Robotium) — Third party Android UI Test Framework ([Robotium vs Espresso](http://stackoverflow.com/a/20487527/5154829))
* [Selendroid](http://selendroid.io/) — Selenium for Android

**Goal of unit testing** - to verify a unit of code (developer code in a method) behaves as expected and that they can handle erroneous input without crashing. A framework JUnit is part of Android API in junit package and android specific testing framework android.test.

To do Unit testing you call a method in you app and test the result for expected output.

This is ASSERTING the result and there is a **class called Assert**

Each test written should perform an assertion on the value, which in turn tells the testing framework whether the test is passed.

Each test is called a test case and a set of tests is test suite

Most methods in your app have dependencies you need a way to isolate a method when running a test and instead use mock objects that simulate the behaviour of dependent objects

mock classes are provided by Android API when using Android build system - default place for all test codes is <project root>/src/instrumentTest

if app name is ken then the test name is ken.test

**What to test? TEST only the code I wrote not Android components**

Write code to test the BUTTON has an onClick listener and that the listener behaves as expected.

AND

Test only one small thing at one time to make it easier to detect a bug and to do refactoring

To TEST your Activity make several small tests - so write tests to enhance overall code design

Refactor your code to extract smaller methods from a single large method

For TESTING Classes that - dont depend on a component lifecycle - use AndroidTestCase class

public class Util {

public static int byteArrayToInt(byte[] bytes) {

return bytes[3] & OxFF |

(bytes[2] & OxFF << 8) |

(bytes[1] &n) OxFF <<16) |

(bytes[0] &n) OxFF << 24;

}

}

the preceding utility class with a method for converting 4 bytes to an integer - no special set up or tear down required

Public class UtilTest extends AndroidTestCase {

.....

}

**Testing Activities**

- means testing all Activity classes use ActivityUnitTestCase and **ActivityInstrumentationTestClass2**

**Example for a code** that shows a simple Activity sets up a View to be displayed along with the method startBackgroundJob() that will used as a click listener in your view

2 things to test in Activity - (1) the existence of a click listener for the Button and (2) that the listener calls Context.startService()

Use either testIfButtonHasClickListener() and testIfClickListenerStartsService() with correct Intent

**Testing Services**

Android testing APIs -> a class - ServiceTestClass

1. create a simple service

2. use simple ServiceTestClass to test it

Here u call setupService() to create all dependencies for Service that is to test

testBinder() tests your bindService()

**Test ContentProviders**

Use ProviderTestCase2<TaskProvider> class

Goal - verify public contract i.e. test if a client receives the data it requests

Also run tests to verify is the database created has expected table and columns

then verify that default values are created correctly

**How to run a test ?**

[1] Use built-in feature of Android Studio

[2] Execute Gradel fromt eh command line with task connectedInstrumentTest

**Continuous Integration**

[1] Use bitbucket and Jira

[2] Use Jenkins CI

**Automated Testing**

* https://software.intel.com/en-us/videos/test-driven-development-and-android-testing-by-christopher-perry-and-marcelo-bay-area-android
* http://www.tivix.com/blog/test-driven-development-android/

**Beginning testing**

* https://developer.android.com/training/testing/start/index.html

**Moving from outside to inside method in TDD**

* https://www.novoda.com/blog/approaching-tdd-outside-in-on-android-i/

**Additional Ideas**

* http://bytes.babbel.com/en/articles/2016-05-20-tdd-in-android.html
* http://stackoverflow.com/questions/6261504/android-test-driven-development
* https://developer.android.com/training/testing/start/index.html

**[1**] **Monkey and monkeyrunner** - to test UI part of app

* https://developer.android.com/training/testing/ui-testing/index.html

[2] Top 5 Testing Frameworks with Examples

* https://developer.android.com/training/testing/ui-testing/index.html

**Monkey MonkeyRunner, etc**.

* https://developer.android.com/studio/test/monkey.html
* https://developer.android.com/studio/test/monkey.html
* https://developer.android.com/studio/test/monkeyrunner/index.html

**List of popular tools**

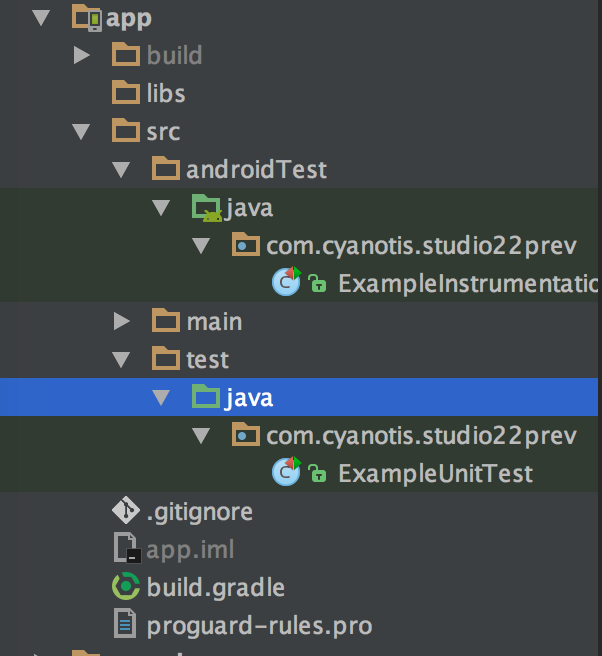
* http://www.softwaretestinghelp.com/5-best-automation-tools-for-testing-android-applications/

[3] **Mockit**o an automated tool for mocking tests

**How to prepare for a testing activity**

**Package Structure for Testing :-**

In our project, two folders host our tests which are ***test*** and ***androidTest.***



**Test :**Unit tests are **hosted in this folder**. These tests run on JVM and do not requires *Android device or emulator*. This type of test have no access to any android framework specific component like *Context*.

**androidTest**:- All Instrumentation (*Espresso*) tests are hosted in this folder. These tests need physical Android device or emulator in order to run.

References

* https://medium.com/mobility/how-to-do-tdd-in-android-90f013d91d7f
* https://medium.com/mobility/how-to-do-tdd-in-android-part-2-project-architecture-setup-unit-testing-77cdd1d6aa3a
* https://developer.android.com/training/testing/ui-testing/index.html
* https://developer.android.com/studio/test/monkey.html
* https://medium.com/mobility/how-to-do-tdd-in-android-part-2-project-architecture-setup-unit-testing-77cdd1d6aa3a
* http://bytes.babbel.com/en/articles/2016-05-20-tdd-in-android.html
* https://developer.android.com/training/testing/start/index.html
* http://blog.carbonfive.com/2012/07/17/beginnin-test-driven-development-in-android/

**[1] Handler Thread**

A Handler on the **UI Thread**. Handler is part of the Android system's framework for managing threads. A Handler object receives messages and runs code to handle the messages. check this out first

* http://android-coding.blogspot.in/2013/10/example-of-using-handlerthread.html and then this
* http://android-er.blogspot.in/2016/03/handlerthread-example.html ...careful this is in Android N and is about multi-pane window

***My Opinion***

I use a background thread to do a job and if I initiate a looper then it gets tied down. How do I get other work done. So I use a handlerthread to spawn multiple sub-threads to handle different tasks

Consider when I connect to wifi - WhatsApp push messages came in, I then paid with NFC at Interac POS at supermarket and then used Wifi Direct or Bluetooth to share on a group a few exciting pictures.

There is one service thread in background - it needs to form a looper for handler to dequeue messages for each of the three channels - NFC, Bluetooth and WhatsApp

When there are many messages there are many pieces in the MessageQueue to be handled and it requires lots of handles - one set for WhatsApp and another set for NFC and another set for Bluetooth

All these handlers are themselves objects and since there are several of them - we can put them in a queue and we now need a worker thread to service this MessageQueue full of handlers or handler objects

The worker thread that will manage to handlers in the MessageQueue is then to dequeue this handler queue and that specialized worker thread is a handlerthread

Note diagram below

Thread 2

Thread 1

Handler Object

Handler Object

Looper

Looper

Diagram 1 - illustrates 2 worker threads and their loopers and handler objects

And alternatively now imagine a different scenario

Diagram 2 - all handler objects are in a queue similar to the message queue on a thread

A thread specialized in managing a queue of handler objects for enqueue and dequeue

A Queue of handler objects

And see the sample code below - hope this sharing is useful to you

Handy class for starting a new thread that has a looper. The looper can then be used to create handler classes. Note that start() must still be called.

**Example -** Problem - To set up a HandlerThread from the GUI thread. Then some time later, when a button is clicked on the GUI, it runs callHello(), which then send a message to a HelloLogger object residing on the non-GUI thread which asynchronously logs "Hello World". I have tried a number of things, some block indefinitely, some never receive the message, etc etc. The code below is more or less as close as I have got, please could someone modify it to work?

public class HandlerThreadExample {

private MyHandlerThread mMyHandlerThread;

private Looper mLooper;

private Handler mHandler;

public HandlerThreadExample(){

mMyHandlerThread = new MyHandlerThread();

mMyHandlerThread.start();

mLooper = mMyHandlerThread.getLooper();

}

public void callHello() {

mHandler.sendEmptyMessage(1);

}

private class MyHandlerThread extends HandlerThread {

private HelloLogger mHelloLogger;

private Handler mHandler;

public MyHandlerThread() {

super("The MyHandlerThread thread", HandlerThread.NORM\_PRIORITY);

}

public void run (){

mHelloLogger = new HelloLogger();

mHandler = new Handler(getLooper()){

public void handleMessage(Message msg){

mHelloLogger.logHello();

}

};

super.run();

}

}

private class HelloLogger {

public HelloLogger (){

}

public void logHello(){

Log.d("HandlerThreadExample", "Hello World");

}

}

}

a possible thought towards a solution

Public class HandlerThreadExample2 {

private static int MSG\_START\_HELLO = 0;

private static int MSG\_HELLO\_COMPLETE = 1;

private HandlerThread ht;

private Handler mHtHandler;

private Handler mUiHandler;

private boolean helloReady = false;

public HandlerThreadExample2(){

ht = new HandlerThread("The new thread");

ht.start();

Log.d(App.TAG, "UI: handler thread started");

mUiHandler = new Handler(){

public void handleMessage(Message msg){

if (msg.what == MSG\_HELLO\_COMPLETE){

Log.d(App.TAG, "UI Thread: received notification of sleep completed ");

helloReady = true; }

}

};

mHtHandler = new Handler(ht.getLooper()){

public void handleMessage (Message msg){

if (msg.what == MSG\_START\_HELLO){

Log.d(App.TAG, "handleMessage " + msg.what + " in " + Thread.currentThread() + " now sleeping");

try {

Thread.sleep(2000);

} catch (InterruptedException e) {

e.printStackTrace();

}

Log.d(App.TAG, "Woke up, notifying UI thread...");

mUiHandler.sendEmptyMessage(MSG\_HELLO\_COMPLETE);

}

}

};

}

public void sendLongHello(){

if (helloReady){

Log.d(App.TAG, "sending hello " + Thread.currentThread());

mHtHandler.sendEmptyMessage(MSG\_START\_HELLO);

helloReady = false;

} else {

Log.e(App.TAG, "Cannot do hello yet - not ready");

}

}

}

Alternatively

HandlerThread ht = new HandlerThread("MySuperAwesomeHandlerThread");

ht.start();

Handler h = new Handler(ht.getLooper()) {

public void handleMessage(Message msg) {

Log.d(TAG, "handleMessage " + msg.what + " in " + Thread.currentThread());

};

};

for (int i = 0; i < 5; i++) {

Log.d(TAG, "sending " + i + " in " + Thread.currentThread());

h.sendEmptyMessageDelayed(i, 3000 + i \* 1000);

}

Also think about this

Make two class fields:

Handler mHtHandler;

Handler mUiHandler;

and try this:

HandlerThread ht = new HandlerThread("MySuperAwsomeHandlerThread");

ht.start();

Callback callback = new Callback() {

@Override

public boolean handleMessage(Message msg) {

if (msg.what == 0) {

Log.d(TAG, "got a meaasage in " + Thread.currentThread() + ", now sleeping... ");

try {

Thread.sleep(2000);

} catch (InterruptedException e) {

e.printStackTrace();

}

Log.d(TAG, "woke up, notifying ui thread...");

mUiHandler.sendEmptyMessage(1);

} else

if (msg.what == 1) {

Log.d(TAG, "got a notification in " + Thread.currentThread());

}

return false;

}

};

mHtHandler = new Handler(ht.getLooper(), callback);

mUiHandler = new Handler(callback);

mHtHandler.sendEmptyMessageDelayed(0, 3000);

References

* http://stackoverflow.com/questions/25094330/example-communicating-with-handlerthread
* https://medium.com/@ali.muzaffar/handlerthreads-and-why-you-should-be-using-them-in-your-android-apps-dc8bf1540341

**[2] Managing Memory in Android, How to find out un-used objects in memory or If such un-used objects exist**

Random-access memory (RAM) is a valuable resource in any software development environment, but it's even more valuable on a mobile operating system where physical memory is often constrained. Although both the Android Runtime (ART) and Dalvik virtual machine perform routine garbage collection, this does not mean you can ignore when and where your app allocates and releases memory. You still need to avoid introducing memory leaks, usually caused by holding onto object references in static member variables, and release any [Reference](https://developer.android.com/reference/java/lang/ref/Reference.html)objects at the appropriate time as defined by lifecycle callbacks.

A **memory leak** happens when allocated memory cannot be freed. When an object is expected to be unused but there is at least one reference to it, GC will fail to recognize it as unused. It stays in memory which makes the available space smaller and thus, triggers the GC again and again.

GC events may affect rendering and cause frame skips if they took longer than expected or if they happen too often. The more GC events, the less time for other operations such as rendering.

When a high amount of objects is allocated and freed in a small amount of time, it is considered as a **memory churn**.

**To check for GC event issues:**

* Use the [**Memory Monitor Tool**](http://developer.android.com/tools/performance/memory-monitor/index.html) which shows how your app is using memory over time.
* How to Launch: Android Studio > Tools > Android > Memory Monitor
* Light blue is free memory or memory available to be used.
* Dark blue is allocated memory or memory being used.
* Every drop/dip in the graph is a garbage collection event. Lots of dips in a short amount of time signal performance problems.
* If there are problems, consider using the Heap tool and the Allocation Tracker tool.

**To detect and fix memory leaks:**

Assume we have an activity that has bitmaps that have to be GCed after the activity is destroyed:

1. Create a blank activity with a low or known memory allocation and create a way for the bitmap activity to transition into it.
2. Use the [**Heap Viewer**](http://developer.android.com/tools/performance/heap-viewer/index.html) to get a snapshot of how memory normally looks like in the activity that has bitmaps.
3. Transition into the blank activity and click “Cause GC” to force garbage collection which should remove allocations from the bitmap activity.
4. Once you’re in the blank activity, do another heap dump and inspect if there are allocations that shouldn’t be there.
5. If there are suspects, switch to the [**Allocation Tracker**](http://developer.android.com/tools/performance/allocation-tracker/index.html) tool to know what allocations are occurring.
6. To use allocation tracker, Press Start Allocation Tracking, open the bitmap activity, transition to the blank activity, press “Cause GC”, then press Stop Allocation Tracking.
7. After a while, the tool will display a list of all objects that were created, the order in which they were created and where they were created.
8. You can now track down the allocations on your code and figure out why they are not being freed.

**To prevent memory churn:**

1. Avoid allocation in inner loops.
2. Avoid onDraw allocations.
3. Consider using an object pool. The pool allocates a group of objects and allows them to be grabbed and returned once usage is done. Objects are pulled from the pool instead of allocating them again on the memory heap.

How to use a Heap Viewer

**Memory Leak Patterns in Android**

**How to Identify a Leak**

* Use tools like
  + Leak Canary from Square
  + In Android Studo there is a method
* Android Studio has a handy tool for detecting memory leaks. If you suspect a piece of code in you app might leaks an Activity, you can do this.
  + Step 1: Compile and run the debug build on a device or emulator connecting to your computer.
  + Step 2: Go to the suspicious activity, then go back to previous activity which will pop the suspicious activity from the task stack.
  + Step 3: In Android Studio -> Android Monitor window -> Memory section, click on Initiate GC button. Then click on Dump Java Heap button.
  + Step 4: When Dump Java Heap button is pressed, Android Studio will open the dumped .hprof file. In the hprof file viewer, there are a couple of ways you can check the memory leak. You can use the Analyzer Tasks tool on the top right corner to detect leaked activities automatically. Or you can switch the view mode to Package Tree View from top left corner, find the activity which should be destroyed. Check the Total Count of the activity object. If there are 1 or more instances, it means there is a leak.
  + Step 5: Once you find the leaked Activity, check the reference tree on the bottom and find out what object is referencing the should-have-been-dead activity.

Example

Build and run a simple app which calls 2 or 3 activities and on click of the home button kills activities 2 an d 3 to come to home screen and then check for leaks using above method. If you can track old activities 2 or 3 exist then there is a memory leak.

* Common 3 Leak Patterns - when there is a leak activity to a static reference; when there is a leak activity to a thread,; when there is a leak thread itself

References

* https://developer.android.com/topic/performance/memory.html
* http://androidnotes.github.io/starters/skims/tools/2016/03/30/android-perf-patterns-skim-1-memory.html#sthash.UAPQpamf.dpuf
* https://www.cs.princeton.edu/courses/archive/spr96/cs333/java/tutorial/java/javaOO/garbagecollection.html
* https://www.slideshare.net/zblair/identifying-memory-leaks-in-android-applications
* https://android.jlelse.eu/memory-leak-patterns-in-android-4741a7fcb570

**[3] Android Launch modes**

Activity is one of the most brilliant concept on Android from its well-design architecture on memory management which lets Multitasking works perfectly on this most popular mobile operating system.

Anyway, Activity is not just to be launched on the screen. The way it is launched is also concerned. There are so many details in this topic. One of those that is really important is **launchMode**.

Since each Activity is made to work in different purpose. Some is designed to work separately with each Intent sent for example an Activity for email composing in email client. While some is designed to work as a singleton for example an email's inbox Activity.

The different Launch Modes

* Standard
* SingleTop - used in a Search function
* singleTask
* singleInstance

Beside from assigning the launch mode directly in AndroidManifest.xml, we are also able to assign more behavior through thing called **Intent Flags**, for example:

Intent intent = new Intent(StandardActivity.this, StandardActivity.class);

intent.addFlags(Intent.FLAG\_ACTIVITY\_SINGLE\_TOP);

startActivity(intent);

would launch a StandardActivity with singleTop launchMode condition.

A task is the stack (“Last in, First out”) which contains a collection of activity instances. Normally, when a user starts an app a new task will be created, and the first activity instance is called as a root of the task. The Android system can hold multiple tasks at the same time and only one task is in the foreground. Similar to a double click of the home key on iOS, if you long press the HOME key on Android you’ll be presented with a list of your currently running applications. You can select any one of these applications (which are currently running in the background) in order to bring it to the foreground and interact with the app! Of course, because background tasks do tend to use up your processor cycles you should try to keep your backgrounded apps to a minimum to ensure your phone performs optimally.

Now let’s look at the launch mode of an activity. Launch mode allows you to define how a new instance or the existing instance of an activity is associated with the current task.

**Example**

|  |  |
| --- | --- |
|  | Between the [Browser](http://android.git.kernel.org/?p=platform/packages/apps/Browser.git;a=blob;f=AndroidManifest.xml;h=3d970ab3984bb06057a6ca756cb59b7738581967;hb=HEAD) and [Alarm Clock](http://android.git.kernel.org/?p=platform/packages/apps/AlarmClock.git;a=blob;f=AndroidManifest.xml;h=d7310097bf2c247314ec1aa38ffaa7da952e6fa0;hb=HEAD) applications, you cover all four launch modes:   1. [**BrowserActivity**](http://android.git.kernel.org/?p=platform/packages/apps/Browser.git;a=blob;f=src/com/android/browser/BrowserActivity.java;h=60dffac9e075748a01d8d761d087bbbcb687b9d0;hb=HEAD) uses **singleTask**. There is only one browser activity at a time and it doesn't become part tasks that send it intents to open web pages. While it might return to whatever most recently launched it when you hit back it is actually fixed at the bottom of its own task activity stack. It will share its task with activities that it launches like bookmarks. 2. [**BrowserBookmarksPage**](http://android.git.kernel.org/?p=platform/packages/apps/Browser.git;a=blob;f=src/com/android/browser/BrowserBookmarksPage.java;h=6ab011bfadd147f430050e174c8b86eb33299c97;hb=HEAD) uses **singleTop**. While there can be multiple instances of this activity, if there is already one at the top of the task's activity stack it will be reused and [onNewIntent()](http://developer.android.com/reference/android/app/Activity.html#onNewIntent%28android.content.Intent%29)will be called. This way you only have to hit back once to return to the browser if the bookmarks activity is started multiple times. 3. [**AlarmClock**](http://android.git.kernel.org/?p=platform/packages/apps/AlarmClock.git;a=blob;f=src/com/android/alarmclock/AlarmClock.java;h=75477fd01568d438c543facb34e8f8c877459611;hb=HEAD) uses **standard**. The user can launch multiple instances of this activity and these instances can be part of any task and anywhere in the activity stack. As a fairly simple application it doesn't really demand tight control of its activity. 4. [**AlarmAlert**](http://android.git.kernel.org/?p=platform/packages/apps/AlarmClock.git;a=blob;f=src/com/android/alarmclock/AlarmAlert.java;h=89a866a991ebecce2a0114f2d5c3f08c378ff826;hb=HEAD) uses **singleInstance**. Only one alert activity at a time and it is always its own task. Anything it launches (if anything) becomes part of its own new task. |

# [Setting launchMode=“singleTask” vs setting activity launchMode=“singleTop”](http://stackoverflow.com/questions/25773928/setting-launchmode-singletask-vs-setting-activity-launchmode-singletop)

We often get confused with Android Activity Launch Mode when we see **android:launchMode** attribute associated with <activity> element in manifest file, it defines the way it will be associated to a task. A task is a collection of activities that users interact with when performing a certain job. Task uses Stack to maintain the history of activities.

Android Activity Launch Mode can be opted from any four modes as described below.

**Standard (Default Launch Mode If not specified) <activity android:launchMode="standard" />**

A new instance of actvity is created everytime in the task from which it was started. Multiple instances of activity can be created and each instance may belong to different task.

**SingleTop <activity android:launchMode="singleTop" />**

If instance of activity is present on top of Task stack, a new instance will not be created and system will route your intent information through onNewIntent(). If it is not presnt on top, a new instance will be created. Multiple instance can be created and each instance may belong to different task.

**SingleTask <activity android:launchMode="singleTask" />**

A new task will be created and a new instance of the activity will be pushed at root of new task. If instance exists on the seperate task then system routes the call to existing instance through onNewIntent() method. Only one instance will exist at a time.

**SingleInstance <activity android:launchMode="singleInstance" />**

Only one instance will exist at a time. System will not launch any other activity into task holding this type. It is always a single member of its task and activities started from here will open into seperate task.

**Sample Android Application to Understand Activity Launch Mode**

Alternatively you can also say

standard (default) :- Multiple instances of the activity class can be instantiated and multiple instances can be added to the same task or different tasks. This is the common mode for most of the activities.

singleTop :- The difference from standard is, if an instance of the activity already exists at the top of the current task and the system routes the intent to this activity, no new instance will be created because it will fire off an onNewIntent() method instead of creating a new object.

singleTask:- A new task will always be created and a new instance will be pushed to the task as the root. However, if any activity instance exists in any tasks, the system routes the intent to that activity instance through the onNewIntent() method call.

In this mode, activity instances can be pushed to the same task. This mode is useful for activities that act as the entry points.

singleInstance:- Same as singleTask, except that the no activities instance can be pushed into the same task of the singleInstance’s.

Accordingly, the activity with launch mode is always in a single activity instance task. This is a very specialized mode and should only be used in applications that are implemented entirely as one activity.

Examples

* http://androidsrc.net/android-activity-launch-mode-example/
* http://stackoverflow.com/questions/25773928/setting-launchmode-singletask-vs-setting-activity-launchmode-singletop
* https://blog.mindorks.com/android-activity-launchmode-explained-cbc6cf996802
* https://www.mobomo.com/2011/06/android-understanding-activity-launchmode/

**Task, Back Task**

A **task** is a collection of activities that users interact with when performing a certain job. The activities are arranged in a stack (the back stack), in the order in which each activity is opened. When apps are running simultaneously in a [multi-windowed environment](https://developer.android.com/guide/topics/ui/multi-window.html), the system manages tasks separately for each window; each window may have multiple tasks. The same holds true for [Android apps running on Chromebooks](https://developer.android.com/topic/arc/index.html): the system manages tasks, or groups of tasks, on a per-window basis.

**The device Home screen is the starting place for most tasks**. When the user touches an icon in the application launcher (or a shortcut on the Home screen), that **application's task comes to the foreground**.

If no task exists for the application (the application has not been used recently), then a new task is created and the "main" activity for that application opens as the root activity in the stack.

When the current activity starts another, the new activity is pushed on the top of the stack and takes focus.

The previous activity remains in the stack, but is stopped. When an activity stops, the system retains the current state of its user interface.

When the user presses the **Back** button, the current activity is popped from the top of the stack (the activity is destroyed) and the previous activity resumes (the previous state of its UI is restored).

Activities in the stack are never rearranged, only pushed and popped from the stack—pushed onto the stack when started by the current activity and popped off when the user leaves it using the **Back** button. As such, the back stack operates as a "last in, first out" object structure.

If the user continues to press **Back**, then each activity in the stack is popped off to reveal the previous one, until the user returns to the Home screen (or to whichever activity was running when the task began). When all activities are removed from the stack, the task no longer exists.

Diagram - to illustrate how activities are kept in the memory on a stack called the back stack

Back Stack -

holds objects

A collection of activities that form a TASK

e.g. multi-window environment

**Managing the Back Stack**

Each activity has activity lifecycle methods you can override to achieve the result you need. Thus, you can either launch Activity2 onResume() on Activity1 onPause(),

<http://developer.android.com/training/basics/activity-lifecycle/index.html>

or, invoke ActivityManager to detect and manage the other activities.

<http://developer.android.com/reference/android/app/ActivityManager.html>

You can also make use of intent resolution mechanism to assign several priorities to your activities and then setup intent filters in each activity so you can start activities with a given priority in your code. You can do this either in Java or XML (though I suggest Java). Have a look at the Intent class.

**Proper Back Navigation**

**Back navigation** is how users move backward through **the history of screens** they previously visited.

All Android devices **provide a Back button** for this type of navigation, so **your app should not add a Back button to the UI**.

In almost all situations, the system maintains a back stack of activities while the user navigates your application.

This allows the system to properly navigate backward when the user presses **the Backbutton**. However, there are a few cases in which your app should manually  specify the Back behavior in order to provide the best user experience.

**Navigation patterns** that require you to manually specify the *Back* behavior include:

* When the user enters a **deep-level activity** directly from a [notification](https://developer.android.com/guide/topics/ui/notifiers/notifications.html), an [app widget](https://developer.android.com/guide/topics/appwidgets/index.html), or the [navigation drawer](https://developer.android.com/training/implementing-navigation/nav-drawer.html).
* Certain cases in which the user navigates between [fragments](https://developer.android.com/guide/components/fragments.html).
* When the user navigates web pages in a [WebView](https://developer.android.com/reference/android/webkit/WebView.html).

How to implement proper***Back* navigation** in these situations is described in the following sections

# [How to resume Fragment from BackStack if exists](http://stackoverflow.com/questions/18305945/how-to-resume-fragment-from-backstack-if-exists)

I am learning how to use fragments. I have three instances of Fragment that are initialized at the top of the class. I am adding the fragment to an activity like this:

Declaring and initializing:

Fragment A = new AFragment();

Fragment B = new BFragment();

Fragment C = new CFragment();

Replacing/Adding:

FragmentTransaction ft = getSupportFragmentManager().beginTransaction();

ft.replace(R.id.content\_frame, A);

ft.addToBackStack(null);

ft.commit();

These snippets are working properly. Every fragment is attached to the activity, and is saved to the back stack without any problem.

So when I launch A, C, and then B, the stack looks like this:

| |

|B|

|C|

|A|

\_\_\_

And when I press the 'back' button, B is destroyed and C is resumed.

But, when I launch fragment A a second time, instead of resuming from back stack, it is added at the top of the back stack

| |

|A|

|C|

|A|

\_\_\_

But I want to resume A and destroy all fragments on top of it (if any). Actually, I just like the default back stack behavior.

**How do I accomplish this?**

Expected: (A should be resumed and top fragments should be destroyed)

| |

| |

| |

|A|

\_\_\_

Sample Code

private void selectItem(int position) {

Fragment problemSearch = null, problemStatistics = null;

FragmentManager manager = getSupportFragmentManager();

FragmentTransaction ft = manager.beginTransaction();

String backStateName = null;

Fragment fragmentName = null;

boolean fragmentPopped = false;

switch (position) {

case 0:

fragmentName = profile;

break;

case 1:

fragmentName = submissionStatistics;

break;

case 2:

fragmentName = solvedProblemLevel;

break;

case 3:

fragmentName = latestSubmissions;

break;

case 4:

fragmentName = CPExercise;

break;

case 5:

Bundle bundle = new Bundle();

bundle.putInt("problem\_no", problemNo);

problemSearch = new ProblemWebView();

problemSearch.setArguments(bundle);

fragmentName = problemSearch;

break;

case 6:

fragmentName = rankList;

break;

case 7:

fragmentName = liveSubmissions;

break;

case 8:

Bundle bundles = new Bundle();

bundles.putInt("problem\_no", problemNo);

problemStatistics = new ProblemStatistics();

problemStatistics.setArguments(bundles);

fragmentName = problemStatistics;

default:

break;

}

backStateName = fragmentName.getClass().getName();

fragmentPopped = manager.popBackStackImmediate(backStateName, 0);

if (!fragmentPopped) {

ft.replace(R.id.content\_frame, fragmentName);

}

ft.setTransition(FragmentTransaction.TRANSIT\_FRAGMENT\_FADE);

ft.addToBackStack(backStateName);

ft.commit();

// I am using drawer layout

mDrawerList.setItemChecked(position, true);

setTitle(title[position]);

mDrawerLayout.closeDrawer(mDrawerList);

}

The problem is, when I launch A and then B, then press 'back', B is removed and A is resumed. and pressing 'back' a second time should exit the app.

But it is showing a blank window and I have to press back a third time to close it.

Also, when I launch A, then B, then C, then B again...

Expected:

| |

| |

|B|

|A|

\_\_\_

Actual:

| |

|B|

|B|

|A|

\_\_\_

Should I override onBackPressed() with any customization or am I missing something?

# [Fragment back-stack management on android](http://stackoverflow.com/questions/29585449/fragment-back-stack-management-on-android)

I have several fragments. They transition using slid-in animations. MainFragment-> Fragment2 -> Fragment3.

1) I want to clear backstack when user reach MainFragment, leaving just that fragment in the stack.

2) In some cases I want to remove the previous item in the stack. Say when user press back button and move from fragment3 = > fragment2.

I have shown my code below. If I use that method, the fragment transitions animations get messed up. I am not sure how to implement the second requirement.

Another question is when I remove fragments from backstack do they get disposed?

private void replace(Fragment fragment )

{

if(fragment == null) return;

FragmentTransaction fragmentTransaction = fragmentManager

.beginTransaction();

fragmentTransaction.setCustomAnimations(R.anim.fragment\_enter,

R.anim.fragment\_exit);

fragmentTransaction.replace(R.id.fragment, fragment);

fragmentTransaction.addToBackStack(fragment.toString());

fragmentTransaction.commit();

if(fragment instanceof MainFragment)

clearBackStack();

}

public void clearBackStack() {

if (fragmentManager.getBackStackEntryCount() > 0) {

FragmentManager.BackStackEntry first = fragmentManager.getBackStackEntryAt(0);

fragmentManager.popBackStack(first.getId(), FragmentManager.POP\_BACK\_STACK\_INCLUSIVE);

}

}

Alternatively

Here I am calling the add method of FragmentTransaction. From Google sample code @ [Fragments](http://developer.android.com/guide/components/fragments.html) and using your code as sample:

FragmentTransaction transaction = getFragmentManager().beginTransaction();

// and add the transaction to the back stack

transaction.add(R.id.fragment, fragment3);

transaction.addToBackStack(fragment3.toString());

// Commit the transaction

transaction.commit();

Note: The **add** method does not replace/remove fragment2 as the previous fragment. Call transaction.add with **fragment3** after using or adding **fragment2**.

Therefore the **back key** will show fragment2 when fragment3 is displayed.

Examples

* https://www.mobomo.com/2011/06/android-understanding-activity-launchmode**/**
* http://stackoverflow.com/questions/20953799/android-activity-backstack-management
* https://developer.android.com/reference/android/app/ActivityManager.html
* http://vinsol.com/blog/2014/09/19/transaction-backstack-and-its-management/

**Conclusion**

This is a do-it yourself collection of notes from Google Android documentation and many different open source tutorials the different links are shown in the reference and illustrated examples sub-sections of each chapter.

As on May 2017 this is a continued effort to refine and make this a better notes collection for the first learner of Android App development and hopes to contribute to the beginner level developer as well as the intermediate level developer

The notes are compiled in the word docx format to allow the learner to add and change and edit each and every section to suit her or his choice and adapt it to the specific work environment.

The author has 20 years of software engineering and development experience and in Android about 9 years of hands-on, code centric focus of contribution in the world market including a few of the biggest brands today.

Many corporate houses have availed of the author's services to coach, train, mentor fresh talent and experienced hire in different countries including USA, Canada, UK, Germany, Singapore, Hong Kong and Australia and in India.

The number of Android devices continue to grow and the market size is nearly 80% of the total number of mobile smart phones globally and the Android fragmentation has created ample market opportunities in Android Wear, Glass, Auto, TV, etc. while adding to the core issue of so many different devices, screen-sizes, versions of Android, and so o.

The author also contributes to iOS 9, 10 development and in Angular 2/ React JS development as well as Big Data with Hadoop, Spark and Scala and is now involved in applying these technologies to the construction industry with BIM + CAFM and to Healthcare, Hospitals, Hotel and Restaurants and International Trade.

It is hoped such a collection of notes will give rise to a few excellent books in a year or two.

As a reader, learner and developer your feedback will be most appreciated.