Android Notes

November 3, 2015

Introduction

Android is an operating system originally designed for touchscreen-based smartphones, but has since been extended for tablets, televisions, smartwatches and other devices as well. Android was originally developed by Android, Inc. Google purchased the company in 2005. The Open Handset Alliance (OHA), consisting of Google, device manufacturers like HTC, Sony and Samsung and various wireless carriers and chipset makers, was formed in 2007 for developing and promoting open standards for mobile computing, including Android. Android is an open source operating system. HTC Dream, launched in October 2008, was the first commercially available Android phone. Since then Android has grown to become one of "big-two" operating systems for smartphones and tablets.

A new version of Android is first developed internally by Google and a Nexus series device by one of the hardware partners is launched with the new version. The Nexus series devices provide the "pure" Android experience. After that, the source code is released under the Android Open Source Project (AOSP). Manufactures take this source code, add device drivers to it, customize it, put their own user interface elements ("skins") on top of it and compile it for their devices.

Structure

Android is based on Linux and uses a modified Linux kernel. Android is a multitasking operating system. Applications for Android are called Apps. Apps are developed using the Java programming language and the Android SDK (Software Development Kit). Android uses the Dalvik virtual machine for executing the Java code. Android devices can have a widr variety of sensors and hardware devices and the apps SDK lets the apps access these. Apps are typically downloaded from an app store, Google Play being the most common of them. Apps can be free, ad-supproted, trial versions, or paid. Availabity of a large number of apps in the app store is a major strength of the Android platform. The apps run in their own sandboxes; each app that is installed gets its own Linux user and runs under that user's previleges. Access to hardware and sharing of data between different applications require explicit permissions.

Key Characteristics

- Android is a complete platform for smartphones; it provides the operating system, the development platform and the SDK, the execution platform and a range of standard services
- It is free and open source
- Supported by the Open Handset Alliance, including Google, device manufacturers like HTC, Sony and Samsung and various wireless carriers and chipset makers
- Based on Linux
- Supports a range of hardware devices, small and large
- Applications are developed on a modified Java platform
- Applications can access a wide variety of hardware and system elements, including telephony, Wi-Fi, text messaging, PIM (Personal Information Management) information like contacts, address book, SMS inbox, etc., camera, Bluetooth, motion sensors, GPS, streaming media, file system, database management system and so on
- The SQLite lightweight relational database management system is used for database management
- OpenGL graphics library is used for 2D and 3D graphics
- Custom applications have almost the same powers as built-in ones
- Custom applications may replace the built-in applications without having to modify the operating system
- Security is enforced. All applications run in sandboxes. Explicit permission from the user is needed to access anything outside the sandbox
- Multitasking and multithreading are supported
- The system manages an application's lifecycle, enabling more efficient battery and memory usage
- Applications (Apps) are typically installed from the App Store (standard Android supports Google Play, formerly known as Google App Store, but customized versions may restrict access only to specific app stores)
- There is support for free, paid, trial version and ad-supported apps
- Android is designed from the beginning to support internationalization

Development Platform

- The Java programming language
- A class library (the Android SDK) that includes some classes from the Java class libraries, but not all; in particular, user interface is reworked and there are no AWT or Swing classes. It also has a large number of classes of its own. It provides the programmer almost complete access to the platform and the hardware
- Uses the Dalvik JIT virtual machine (said to be optimized for performance on mobile devices) instead of the JVM
  - Java .class files can be compiled into Dalvik .dex executables using the dx tool, one dex file may contain multiple classes
- Starting with Android 4.4 (KitKat), A new experimental AOT (Ahead-of-Time) compiler and runtime known as ART (Android Runtime) was provided as a developer option on an experimental basis. ART is the default runtime starting with Android 5.0 Lollipop. ART Compiles the Dalvik byte code to native code at the time of installation, providing a boost in performance
- Applications are packaged into APK (Android Package) format
- Applications can access a wide variety of hardware and system elements
- The SQLite lightweight relational database management system is used for database management
- Event-driven programming environment
- Seperation between user interface and application logic
- Applications have little control over their life cycle; it is managed by the system for optimal use in a resource-constrained environment. An application that is not interacting with the user at the moment may be terminated anytime without notification. When the user brings the application back to the foreground, it must provide an illusion that it was always running and was never terminated. This requires storing and restoring the state of the application on appropriate events

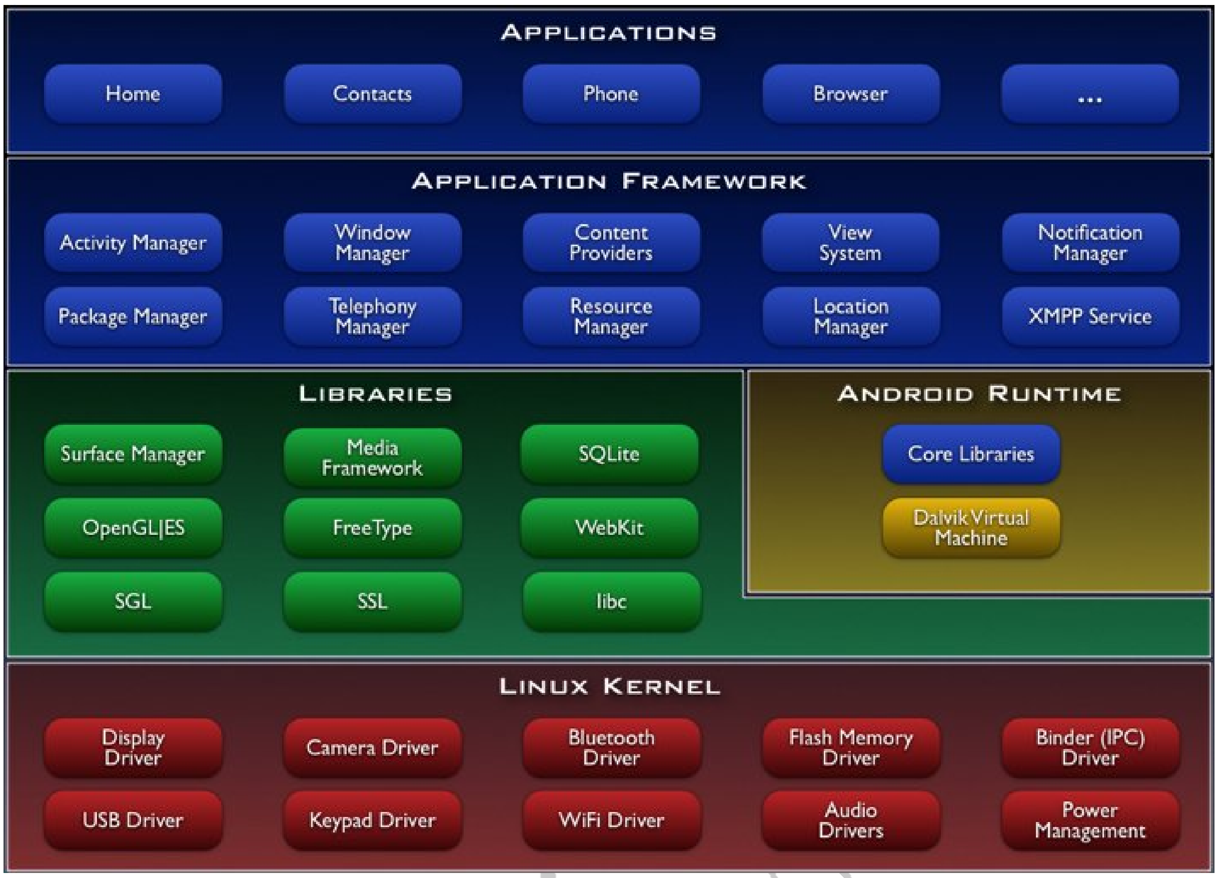


Figure 1: Android System Architecture

Android Versions

Android Version Number	Android Version Name	API Level
Android beta		
Android 1.0		1
Android 1.1		2
Android 1.5	Cupcake	3
Android 1.6	Donut	4
Android 2.0	Eclair	5
Android 2.0.1	Eclair	6
Android 2.1	Eclair	7
Android 2.2–2.2.3	Froyo	8
Android 2.3–2.3.2	Gingerbread	9
Android 2.3.3–2.3.7	Gingerbread	10
Android 3.0	Honeycomb	11
Android 3.1	Honeycomb	12
Android 3.2	Honeycomb	13
Android 4.0–4.0.2	Ice Cream Sandwich	14
Android 4.0.3–4.0.4	Ice Cream Sandwich	15
Android 4.1	Jelly Bean	16
Android 4.2	Jelly Bean	17

Android Version Number	Android Version Name	API Level
Android 4.3	Jelly Bean	18
Android 4.4	KitKat	19
Android 4.4w	KitKat with wearable extensions	20
Android 5.0	Lollipop	21
Android 5.1	Lollipop	22
Android 6.0	Marshmellow	23

Android Logo

Figure 2: The Android Logo

Developing for Android

Eclipse

- Eclipse was the IDE commonly used to develop Android applications. We need to install the classic or Java (SE or EE) edition of Eclipse
- In Eclipse, we need to install the Android Development Tool (ADT) plug-in
- It contains
  - An Android Project Wizard that simplifies creating new projects and includes a basic application template
  - Forms-based manifest, layout, and resource editors to help create, edit, and validate your XML resources
  - Automated building of Android projects, conversion to Android executables (.dex), packaging to package files (.apk), and installation of packages onto Android virtual machines
  - The Android Emulator, including control of the emulator's appearance and network connection settings, and the ability to simulate incoming calls, SMS messages, GPS location, etc
  - The Android Virtual Device (AVD) manager, which lets you create and manage virtual devices that run a specific release of the Android OS and have specific characteristics

  - The Dalvik Debug Monitoring Service (DDMS) and the Android Debug Bridge (ADB) provide complete access to the device (real or emulator) during debugging. The facilities provided include port forwarding, stack, heap, and thread viewing, process details, screen-capture facilities, access to the device or emulator's file system enabling you to navigate the folder tree and transfer files, all Android/Dalvik log and console outputs, etc
  - Android SDK tools
    - SDK Manager, which is used to install different SDK versions and other packages
  - We need at least one (preferably the latest) version of the Android platform SDK. With a newer SDK, we can create apps targeting an older version, but the converse is not true
  - Along with the SDK, we may also get
    - The Android API class library
    - Full documentation
    - Sample code
  - We may obtain further online resources from http://developer.android.com

  The ADT Bundle: Google provides ADT bundles that include everything we need for Android development in a convenient pre-configured package

Android Studio

- Google has developed the Android Studio IDE specifically designed for Android development
- It is based on the IntelliJ IDEA platform (an IDE for Java)
- It is now the default application development tool for Android

Android Application Lifecycle

An Android application may consists of a number of activities. Each activity is a single user interface screen. One activity may invoke another activity. Android maintains an Activity Stack. The activity that the user is currently interacting with is always at the top of the Activity Stack. When that activity starts another activity, the newly started activity is pushed on the top of the stack. When the newly started activity finishes, it is popped from the Activity Stack, bringing the previous activity to the top of the stack again.

The lifecycle of activities is managed by the Android system itself. To conserve scarse resources, especially battery power and memory, Android may terminate a running activity at any time. The activity is supposed to save its state when an event singalling the possibility of such termination occurs. When the activity is restarted, it is passed the previously saved state as a Bundle. The activity should use it to restore itself such

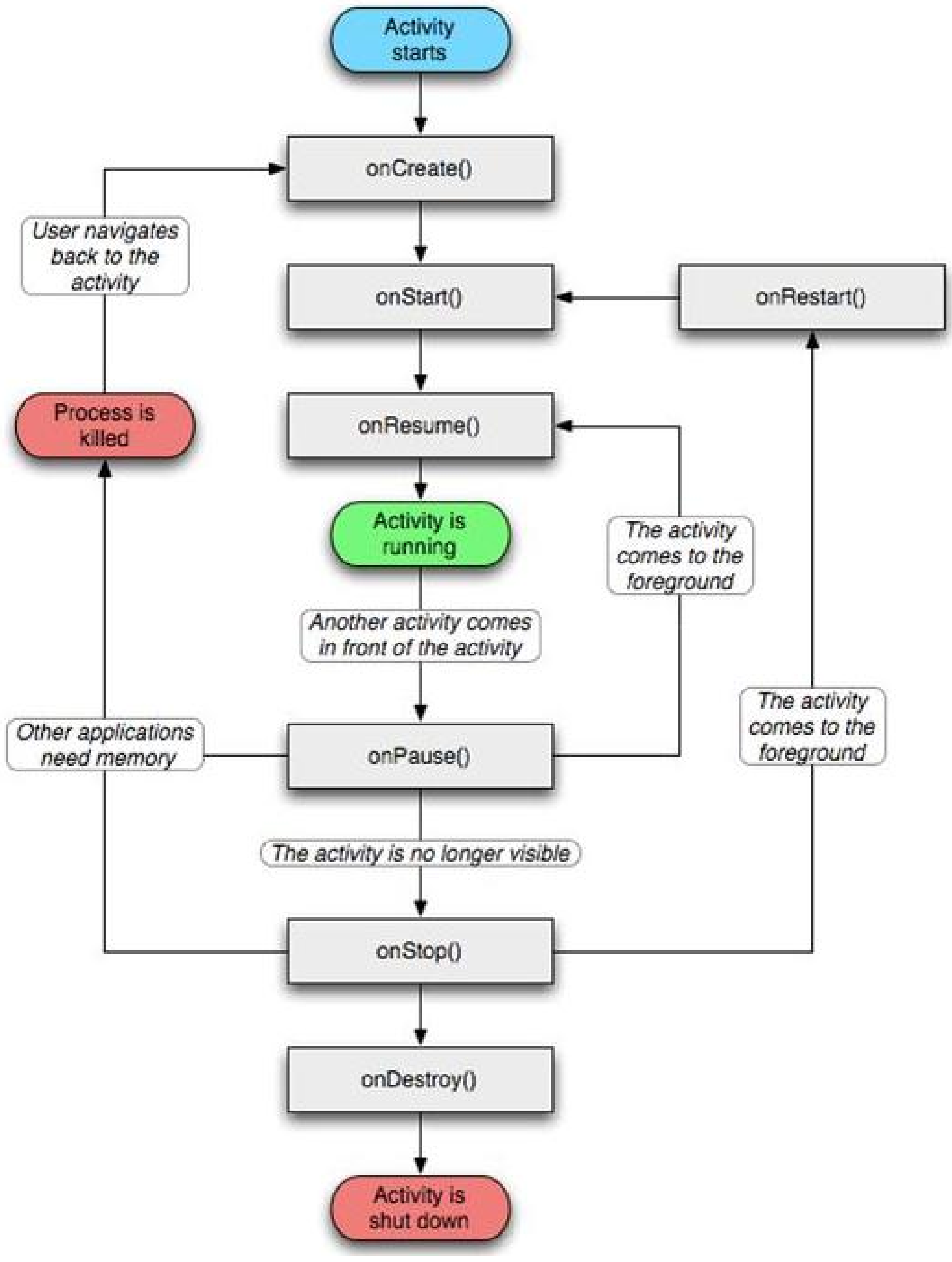
that the user has a seamless experience and does not realize that the activity was terminated and restarted in-between. An activity may also be terminated and restarted when a configuration change, especially a change in the screen orientation or the locale, occurs.
At any time, an activity may be in one of the following states.

- **Resumed (Running)** An activity in this state is currently interacting with the user and has user focus in it. It is on the top of the Activity Stack. Such activity must continue running and update the UI and accept user input when needed, so it is never killed automatically by the Android system
- **Paused** An activity in this state is not currently interacting with the user and does not have user focus in it. It is not on the top of the Activity Stack. However, becauase the top-of-the-stack activity does not occupy the whole screen, this activity is at least partially visible. It is desirable to keep such an activity running and updating its UI, so such activity is killed automatically by the Android system only as a last resort
- **Stopped** An activity in this state is completely obscured by another activity (hidden behind that activity). While it may need to preserve its state (e.g. display), there is no need to waste CPU cycles on keeping it running. Such activity may be killed automatically by the Android system at any time

There are three loops in the lifecycle of an activity.

- **The entire lifetime** of an activity happens between the first call to onCreate(Bundle) through to a single final call to onDestroy(). An activity will do all setup of "global" state in onCreate(), and release all remaining resources in onDestroy(). For example, if it has a thread running in the background to download data from the network, it may create that thread in onCreate() and then stop the thread in onDestroy()
- **The visible lifetime** of an activity happens between a call to onStart() until a corresponding call to onStop(). During this time the user can see the activity on-screen, though it may not be in the foreground and interacting with the user. Between these two methods you can maintain resources that are needed to show the activity to the user. For example, you can register a BroadcastReceiver in onStart() to monitor for changes that impact your UI, and unregister it in onStop() when the user no longer sees what you are displaying. The onStart() and onStop() methods can be called multiple times, as the activity becomes visible and hidden to the user
- **The foreground lifetime** of an activity happens between a call to onResume() until a corresponding call to onPause(). During this time the activity is in front of all other activities and interacting with the user. An activity can frequently go between the resumed and paused states - for example when the device goes to sleep, when an activity result is delivered, when a new intent is delivered - so the code in these methods should be fairly lightweight

Android Activity Lifecycle



Activity starts
onCreate()
User navigates back to the activity
onStart()
onRestart()
Process is killed
onResume()
Activity is running
The activity comes to the foreground
Another activity comes in front of the activity
Other applications need memory
onPause()
The activity comes to the foreground
The activity is no longer visible
onStop()
onDestroy()
Activity is shut down

Figure 3: Android Activity Lifecycle

Android Activity Lifecycle Events

Method	Description	Activity Killable after this Event Occurs?	Possible Next Event
onCreate()	Called when the activity is first created. This is where you should do all of your normal static set up — create views, bind data to lists, and so on. This method is passed a Bundle object containing the activity's previous state, if that state was captured. Always followed by onStart().	No	onStart()
onRestart()	Called after the activity has been stopped, just prior to it being started again. Always followed by onStart()	No	onStart()
onStart()	Called just before the activity becomes visible to the user. Followed by onResume() if the activity comes to the foreground, or onStop() if it becomes hidden.	No	onResume() or onStop()
onResume()	Called just before the activity starts interacting with the user. At this point the activity is at the top of the activity stack, with user input going to it. Always followed by onPause().	No	onPause()
onPause()	Called when the system is about to start resuming another activity. This method is typically used to commit unsaved changes to persistent data, stop animations and other things that may be consuming CPU, and so on. It should do whatever it does very quickly, because the next activity will not be resumed until it returns. Followed either by onResume() if the activity returns back to the front, or by onStop() if it becomes invisible to the user.	Yes	onResume() or onStop()
onStop()	Called when the activity is no longer visible to the user. This may happen because it is being destroyed, or because another activity (either an existing one or a new one) has been resumed and is covering it. Followed either by onRestart() if the activity is coming back to interact with the user, or by onDestroy() if this activity is going away.	Yes	onRestart() or onDestroy()

Method	Description	Activity Killable after this Event Occurs?	Possible Next Event
onDestroy()	Called before the activity is destroyed. This is the final call that the activity will receive. It could be called either because the activity is finishing (someone called finish() on it), or because the system is temporarily destroying this instance of the activity to save space. You can distinguish between these two scenarios with the isFinishing() method.	Yes	None

Saving and Restoring Activity State

Activity state is saved in a Bundle, which implements the Parcelable interface. Items are stored in the bundle as key-value pairs. The interface provides methods like putInt(), putChar(), putFloat() and putString() to put items into the bundle and methods like getInt(), getChar(), getFloat() and getString() to get items from the bundle.

Activity state should be saved when onSaveInstanceState() is called. The method is passed a bundle in which to put the state. Android takes care of storing the bundle and passing it back to the activity when it is recreated. When an activity is created, the bundle is passed to both the onCreate() and onRestoreInstanceState() methods. The activity state can be restored in either of these events. If the passed bundle is null, it means the activity is starting afresh.

Key Concepts in Android Development

- **Android Virtual Device (AVD)** An AVD is an emulator used to emulate specific hardware & software configuration. To test our application against different hardware/software platforms, we may create multiple AVDs

- **Activities** Activities represent the application's presentation tier. Each activity represents a screen that displays some information and may let the user interact with it

- **Views** Views are the basic building blocks of the user interface. They represent (visible or invisible) items that are part of the configuration of the display. Views are the equivalent of GUI components (Java) and controls (.NET). Examples of views are EditText, ListView, etc. that display information in different forms and may let the user interact with it. Views can be nested (one view inside another). A ViewGroup is a special type of view used to group several (sub)views togather. Layouts are (invisible) ViewGroups

- **Services** A Service is a task (process) that runs in the background i.e. without any kind of user interaction. They are used for performing functions that are to be performed periodically without the user invoking the application or for taking some action when particular event occurs

- **Resources** Android follows the philosophy of keeping application logic seperate from the user interface. Hence everything except the code is defined as resources seperate from the code
  - **Different Types of Resources**
    - **Drawables** Drawables are graphics resources that can be painted on the screen. They include icons and images. To cater to different screen sizes, we may have folders like res/drawable-ldpi, res/drawable-mdpi, res/drawable-hdpi, res/drawable-xhdpi, res/drawable-xxhdpi, etc. corresponding to screens with low, medium, high, extra-high and extra-extra-high DPI (Dots Per Inch) screens respectively. Examples of these types of screens would be a small-screen mobile phone, a large-screen mobile phone, a phablet or a 7" tablet, a large tablet and a TV screen respectively. The Android baseline is mdpi. If resources for some screen size are not provided, the nearest-sized resources are automatically scaled to the required sizes by Android. However, this may affect performance and leave visible artefacts on the screen. Having different sized icons and images for different sized screens ensures that the application has nice looks on all types of devices.
    - **Layouts** Layouts are ViewGroups that are not rendered (displayed) themselves, but are used to hold and organize other views geometrically on the screen. layouts decide the sizes and arrangements of the views on screen at runtime. Because there may be a wide variation in screen sizes between different devices, it is not recommended to use fixed sizes and positioning. The typical way of constructing user interface screens is to define the layout in a layout file in XML format and then generate the necessary views at runtime from the XML definition. this process is called layout inflation. Like drawables, we may define diferent layouts for different screen sizes and Android will pick the most suitable one based on the actual screen size at runtime. We may also create views programmatically at runtime without defining them in an XML file. Layouts can be nested one inside another to create a complex layout
    - **Menus** The menus to be used with an activity are defined seperately from the code in XML files. The reasons for doing this are two-fold - to keep language specific menu titles (or images) seperate for easy changeability and to design seperate menus to go with seperate screen layouts
    - **Values** There are different types of values that are stored in XML files seperately from the code for easy changeability. They are:
      - **Dimensions** sizes of various items. We may have different values for different screen sizes
      - **Strings** No string should be hard-coded anywhere in the application code or other resource files (e.g. layouts). All such strings, including labels for view on the screen, titles, menu item titles, etc. should be defined in the strings XML file. This enables internationalization, where the strings may be translated into different languages by non-developers or third parties without going through or having access to

the source code. The systems picks the correct strings XML file at runtime based on the locale (language) settings in effect on the use's device

  · **Styles** Styles are used to define visual appearance of view. They work very similarly to CSS (Cascading Stylesheets). To ensure maximum portability across different devices, one should never define any visual characterisitcs anywhere other than in styles. One style may inherit from another style. There are several built-in styles and a programmer may define one's own style as well. A style applied at the entire application level is called a theme. Themes are used to ensure consistent look and feel throughout the application. They also provide a central place for the styles for easy changeability and for avoiding duplication

  - **Advantages of using Resources**
    * Separating the presentation from the logic
    * Providing different screen layouts, menus and graphic resources (e.g. images/icons) for different screen sizes
    * Internationalization

- **Activity** An Android application may consists of a number of activities. Each activity is a single user interface screen. One activity may invoke another activity. Android maintains an Activity Stack. The activity that the user is currently interacting with is always at the top of the Activity Stack. When that activity starts another activity, the newly started activity is pushed on the top of the stack. When the newly started activity finishes, it is popped from the Activity Stack, bringing the previous activity to the top of the stack again. The lifecycle of activities is managed by the Android system itself. To conserve scarse resources, especially battery power and memory, Android may terminate a running activity at any time. The activity is supposed to save its state when an event singalling the possibility of such termination occurs. When the activity is restarted, it is passed the previously saved state as a Bundle. The activity should use it to restore itself such that the user has a seamless experience and does not realize that the activity was terminated and restarted in-between. An activity may also be terminated and restarted when a configuration change, especially a change in the screen orientation or the locale, occurs

- **Units of dimensions** Android supports a variety of units for specifying the dimensions of various elements. Some of them are:
  - **px** pixels ptpoints (1 point = 1/72 of an inch)
  - **mm** millimeters
  - **in** inches
  - **dp** dp (density-independent pixels) is the recommended unit for specifying dimensions because it considers the screen density (dots per inch) and scales the dimensions accordingly. On different Android devices with very different screen densities, this unit preserves the overall appearance and proportions

- **Notifications** Notifications are a user notification framework. Notifications let you signal users without stealing focus or interrupting their current Activities. They're the preferred technique for getting a user's attention when your activity does not have the focus, say, from within a service or broadcast Receiver. For example, when a device receives a text message or misses an incoming call, it alerts you by flashing lights, making sounds, displaying icons, or showing messages. There is a notification pull-down where these notifications are stored until the user acts on them or dismisses them. On newer versions, the notifications in the notification pull-down may also include images or actionable elements, like buttons

- **Home Screens** These are user-customizable screens. They are the Android equivalent of desktops or workspaces on the personal computers. When no activity is visible on the screen, one of the home screens is on display. The number of available home screens may be different on different devices. Users may decide the appearance of the home screen and place application shortcuts and widgets on them for quick access and prominent display

- **Application Widgets** Application Widgets are small visual application components that can be added to the home screen. Application Widgets can be used to create small dynamic, interactive application components for embedding on the home screen.
  - **Note** The term widget is used to mean user interaction elements (views) in the API reference, while the application widgets are called AppWidgets
  - **Uses of AppWidgets** AppWidgets are used for different purposes:
    * To invoke an activity when tapped
    * To display information that is automatically updated from a background service (e.g. clock, whether report, etc.)
    * To provide controls for the phone (e.g. to mute/unmute phone, to enable/disable Wi-Fi/Bluetooth, etc.)

- **Intents** An intent is an abstract description of an operation to be performed. Intents are messages used for communication between components (like activities, services, broadcast receivers, etc) of the same app or different apps. One may use an explicit intent to send the message to a specific target activity or service or use an implicit intent to broadcast a message (of the operation to be performed) system-wide. In the former case, the system will hand over the intent to the specified activity or service for taking appropriate action while in the latter case, the system will determine the target activity or service that will perform the action as per an algorithm. Intents can be used to invoke activities and services from other apps as well as system apps
  - **Intent Filters** An activity, service or broadcast receiver may define one or more intent filters in its respective entry in the Android manifest file. If one of these components has intent filters declared, it makes that component potentially eligible for receiving implicit intents and broadcase messages that match the criteria in the intent filter (appropriate permission may be required). No intent filters are required for receiving an explicit intent. One may use intent filters to declare interest in recieving messages of a particular type from components of the same app or other apps. One may also use them to substitute

one's own application in place of the system's default application for handling certain types of events (like, for handling incoming text message or email)

  - **Intent filter criteria** Intents are filtered based on several criteria. Some key criteria are as follows:
  - **Category** The category CATEGORY_LAUNCHER indicates that the activity is the main (home/first level) activity in the application. The activity appears in the list of apps and its shortcut may be placed on the home screen
  - **Action** An action indicates the action desired to be performed. There are several built-in actions that are extensively used by the system as well as apps. For example, ACTION_VIEW is used to display some content, ACTION_EDIT is used to initiate editing of some content, ACTION_DIAL is used to dial (call) a number, etc. App developers can also define their own actions
  - **Data** Contains additional data about the action to be performed. Data has two parts:
    * **URI** A URI that points to the data item to be manipulated
    * **MIME type** A MIME (Multimedia Internet Mail Extensions) type that indicated the type of the data item to be manipulated; e.g. text/plain (plain text), text/html (HTML), image/jpeg (JPEG image), image/png (PNG image), etc. There are standard and de-facto standard MIME types for a wide variety of content types

- **Brodcast Messages** Usually a message is passed on to any one target activity or service. In the case of an explicit intent, the intent itself specifies the class of the target. In case of an implicit intent, the Android system determines one target for delivering the message using an algorithm. But in case of a broadcast message, the message is delivered to all recepients that have registered for recieving that type of messages one after anotther. Typically, the Android system generates broadcast messages for hardware and software events in which multiple apps may e interested, for example, a change in network state, a change in location, an incoming text message, etc. User apps can aldo generate broadcast messages. Broadcast messages also use intents as the vehicle for carrying the message

- **Broadcast Receivers** Broadcast Receivers are consumers of broadcast messages. If one creates a broadcast receiver in an application and registers it with the system, it will listen for system-wide broadcast Intents that match specified filter criteria. If a matching intent is fired and the broadcast receiver is selected by the Android algorithm for receiving it, the application will be invoked to respond to the Intent

- **Content Providers** Content Providers are potentially shareable data stores. Content Providers are used to manage and share application databases. They expose content through content:// URIs. They are the preferred means of sharing data across application boundaries. One can configure one's own content providers to access data in a structured way or to give other applications access to one's data. One can also use the content providers exposed by other applications to access their data. Android devices include several native content providers that expose useful data like the media store (for accessing the media on the device like images, videos, music, etc.) and the contacts content provider that provides access to the device user's contacts

- **Cursors** Cursors are scrollable result sets typically generated as a result of executing a SELECT query
- **ListView** A ListView is a scrollable list consisting of 0 or more list items. Each list item may have one or more views (like TextView, ImageButton, CheckBox, etc.) in it as per its layout. Typically, a ListView is populated with list items dynamically using an adapter
- **Adapters** An adapter takes data in a model (an arraylist, a cursor, etc.), generates list items consisting of views (as per list item layout), sets the data from the model in those views and populates a ListView with those list items. Later, when the data in the data model change, the programmer should notify the adapter of the change and the adapter will automatically update the ListView in synch with the modified data. The generation of views can be customized
- **Logging** When an activity is debugged from a PC, an extensive log of events that happen in the emulator and ADB is available in the LogCat window. We can put our own messages to the log using methods like Log.i(TAG, message). These logs messages are a very important mechanism for debugging an app
- **Toast** When an activity, typically running in the foreground, needs to display a short transient message to the mbile user, it uses Toast. The message is displayed as floating text for some duration and then disappears
- **Context** Context provides global information about an application environment and access to application-specific resources and classes, as well as application-level operations such as launching activities, broadcasting and receiving intents, etc. A context object is needed whenever the code needs to access some system service or the user interface
- **Android Manifest File** This is an XML file associated with the application. It specifies several key pieces of information about the application like:
  - Package name
  - Application version number
  - The id of the application icon
  - Information about which activity is the main activity in the application
  - The minimum SDK version required to run the application
  - The SDK version for which the application has been compiled
  - The application name
  - The theme used by the application
  - The activities in the application and the intent filters for the intents they would like to receive
  - The content providers in the application
  - The services in the application
  - The broadcast receivers in the application and the intent filters for the messages they would like to receive

  - The permissions required by the application

- **The UI Thread** In Android, the User Interface (UI) thread, also known as the Event Dispatch Thread (EDT), handles all the user interface events. UI events, as they occur, are added to the event queue. The UI threads retrieves the events from the event queue one by one and invokes the corresponding event handler on the same (UI) threads. Hence, the event handlers must return very quickly, else the UI thread will block waiting for the event handler to return. No UI event in the application can be processed while the event thread is blocked, causing the application to become unresponsive. Android has time limits in which an event handler must return. If the time limit is breached, a dialog is generated by the system informing the user that the application has beome unresponsive and offering the choices of terminating it or waiting for it to respond. This dialog is called the ANR (Application Not Responding) dialog and should never appear in a well-written application. It is imperative that all event handlers always return quickly and any long-running operation is performed in a seperate thread

- **LoaderManager and CursorLoader** The LoaderManager and CursorLoader classes provide the ability of running a database query in the background and updating the user interface when the query completes. The cursor is automatically requeried and the UI updated when there is a change in the database that may affect the query. The LoaderManager class also handles the activity lifecycle by automatically requerying the cursor when there is a configuration change

Some Commonly Used Views

- **TextView** An uneditable view for displaying text
- **EditText** An editable view for displaying and editing text
- **ImageView** A view that displays an image
- **CheckBox** A box that can be checked or unchecked
- **RadioGroup** A group of radio buttons
- **RadioButton** A button that may be selected or unselected. Out of all the radio buttons in a RadioGroup, atmost one button can be in selected state at any point in time. Thus, selecting one radio button automatically unselects all other in the same group
- **ToggleButton** A button that may be in either on or off state at any point in time
- **Spinner** A dropdown list of items from which one item is selected
- **Button** A button that can be clicked to invoke an event handler
- **ImageButton** A button with an image on it instead of text
- **ListView** A view holding a scrollable list of list items
- **DatePicker** A view used for input of a date

- **TimePicker** A view used for input of a time value
- **ScrollView** An invisible ViewGroup that permits scrolling of its contents when necessary. It can have exactly one child view, but the child view can be a ViewGroup or a Layout holding many views

Some Commonly Used Layouts

- **LinearLayout (Vertical)** A LinearLayout with vertical orientation arranges all child views vertically one below another
- **LinearLayout (Horizontal)** A LinearLayout with horizontal orientation arranges all child views horizontally one after another
- **RelativeLayout** A layout that arranges one child view in relation to (e.g. below) another child view
- **TableLayout** A layout that consists of vertically organized TableRow objects. Each TableRow consists of horizontally organized views (cells). The height and width of the cells are adjusted automatically to make the overall layout look like a table, with each column having equal width

Some Commonly used Android Pernissions

With Android, to access hardware (sensors), network, or to perform certain sensitive tasks, an app needs to request permission in its manifest file as follows:

```
<uses-permission android:name="android.permission.SEND_SMS" />
<uses-permission android:name="android.permission.RECEIVE_SMS" />
```

Permission	Use
INTERNET	Allows the use of the local network as well as the Internet
ACCESS_COARSE_LOCATION	Obtain rough location from the network provider
ACCESS_FINE_LOCATION	Obtain precise location using GPS (and A-GPS, if its is availal
ACCESS_NETWORK_STATE	Allows applications to access information about networks
CAMERA	Required to be able to access the camera device.
GET_ACCOUNTS	Allows access to the list of accounts in the Accounts Service.
BLUETOOTH	Allows applications to connect to paired bluetooth devices.
CAMERA	Required to be able to access the camera device.
FLASHLIGHT	Allows access to the flashlight.
READ_CONTACTS	Allows an application to read the user's contacts data.
READ_EXTERNAL_STORAGE	Allows an application to read from external storage.
READ_SMS	Allows an application to read SMS messages.
SEND_SMS	Allows an application to send SMS messages.
WRITE_EXTERNAL_STORAGE	Allows an application to write to external storage.

Annotated Code Snippets

Basic Activity Structure

```
public class MainActivity extends Activity
{

    @Override
    protected void onCreate(Bundle savedInstanceState)
    {
    // With many Android event handlers, one must call the parent class handler
    // before (or, with some events, after) one’s own code
        super.onCreate(savedInstanceState);
    // Inflate the layout
        setContentView(R.layout.activity_main);
    }
    @Override
    public boolean onCreateOptionsMenu(Menu menu)
    {
    // Inflate the menu; this adds items to the action bar if it is present.
        getMenuInflater().inflate(R.menu.main, menu);
        return true;
    }
    @Override
    public boolean onOptionsItemSelected(MenuItem item)
    {
        return super.onOptionsItemSelected(item);
    }

}
```

Using different views (TextView example)

```
public TextView textViewMessage;
...
// In onCreate()
...
textViewMessage = (TextView)findViewById(R.id.textViewMessage);
...
textViewMessage.setText("Welcome ");
```

Displaying a Toast

```
// Don’t forget to show the created Toast!
Toast.makeText(context, "Settings not implemented yet",
                                        Toast.LENGTH_LONG).show();
```

Using a Button

```
public Button buttonOK;
...
// In onCreate()...
buttonOK = (Button)findViewById(R.id.buttonOK);
buttonOK.setOnClickListener(new View.OnClickListener()
{

    @Override
    public void onClick(View view)
    {
        ...
        ...
    }

});
```

Using menus

```
@Override
public boolean onCreateOptionsMenu(Menu menu)
{
    // Inflate the menu; this adds items to the action bar if it is present.
    getMenuInflater().inflate(R.menu.main, menu);
    return true;
}
@Override
public boolean onOptionsItemSelected(MenuItem item)
{

    int menuId = item.getItemId();
    switch(menuId)
        {
            case R.id.menu_1:
              ...
              break;
            case R.id.menu_2:
              ...
              break;
        }
    return super.onOptionsItemSelected(item);

}
```

Shwoing and Hiding a View

```
textViewEmail.setVisible(View.VISIBLE);
textViewEmail.setVisible(View.GONE);
```

Enabling and Disabling a View

```
textViewEmail.setEnabled(true);
textViewEmail.setEnabled(false);
```

Using the Context Menu

```
// in onCreate()
registerForContextMenu(listViewContactName);
@Override
public void onCreateContextMenu(ContextMenu menu, View v,
                                ContextMenuInfo menuInfo)
{
getMenuInflater().inflate(R.menu.contact_list_context_menu, menu);
super.onCreateContextMenu(menu, v, menuInfo);
}
@Override
public boolean onContextItemSelected(MenuItem item)
{
AdapterContextMenuInfo info = (AdapterContextMenuInfo) item.getMenuInfo();
int itemId = item.getItemId();
switch(itemId)
{
case R.id.edit_contact:
...
break;
case R.id.delete_contact:
...
break;
}
return super.onContextItemSelected(item);
}
```

Saving and Restoring Instance State

```
@Override
protected void onSaveInstanceState(Bundle outState)
{
// The contents of views with unique id
// and user-editability are saved and restored
// automatically by Android
// No need to save and restore their state in our code
// Just call the super class method
outState.putString("editText3", editText3.getText().toString());
outState.putString("textViewData", textViewData.getText().toString());
super.onSaveInstanceState(outState);
}
@Override protected void onRestoreInstanceState(Bundle savedInstanceState)
{
```

```
editText3.setText(savedInstanceState.getString("editText3"));
textViewData.setText(savedInstanceState.getString("textViewData"));
// Don't forget to call the super class method
super.onRestoreInstanceState(savedInstanceState);
}
```

Creating a Confirmation Dialog

```
private void confirm(String message, String okButtonText, String cancelButtonText
{
Context context = this;
    String title = "Confirmation";
    final AlertDialog.Builder ad = new AlertDialog.Builder(context);
     ad.setTitle(title);
    ad.setMessage(message);
//     Use ad.setPositiveButton() to set the "Positive" (OK or YES) button
//     Use ad.setNegativeButton() to set the "Negative" (Don't or No) button
//
To permit dismissal of the dialog without pressing a button (using back button)
//  ad.setCancelable(true);
//
Use ad.setOnCancelListener() to set the action to be taken when
//                              the dialog is cancelled
// Don't forget to show the created dialog!
ad.show();
}
```

Using an ArrayAdapter

```
public static ArrayList<String> contacts;
public static ArrayAdapter<String> adapterContacts;
contacts = new ArrayList<String>();
adapterContacts =
        new ArrayAdapter<String>(context,
             android.R.layout.simple_list_item_1, contacts);
listViewContactName.setAdapter(adapterContacts);
contacts.add("aaa");
contacts.add("bbb");
adapterContacts.notifyDataSetChanged();
```

Invoking a Fixed Activity Using Intent

```
Intent editContactDetails = new Intent(context, ContactDetails.class);
    // If activity result is needed
startActivityForResult(editContactDetails, 1);
  // Result would be Activity.RESULT_OK or Activity.RESULT_CANCELED
// If result is not needed
// startActivity(editContactDetails);
// To process activity result
@Override
protected void onActivityResult(int requestCode, int resultCode, Intent data)
{
    super.onActivityResult(requestCode, resultCode, data);
    switch(requestCode)
    {
         case (1):
             if (resultCode == Activity.RESULT_OK)
             ...
             else if (resultCode == Activity.RESULT_CANCELED)
                    ...
             break;
    }
}
```

Communicating Data Through Intent

```
Intent editContactDetails = new Intent(context, ContactDetails.class);
// Optionally pass additional information through the intent
editContactDetails.putExtra("action", 1);
editContactDetails.putExtra("selectedContactName", “abc”);
// retreiving extra data from intent
// In ContactDetails class
// If the key is not found, -1 is returned
int action = getIntent().getIntExtra("action", -1);
// If the key is not found, null is returned
selectedContactName = getIntent().getStringExtra("selectedContactName");
```

The DatabaseOpenHelper Class

```
public class ContactDatabaseOpenHelper extends SQLiteOpenHelper
{
      private static final String DATABASE_NAME = "contacts.db";
      private static final int DATABASE_VERSION = 1;
// Database creation sql statement
      private static final String DATABASE_CREATE =
                      "create table contacts (
                       id integer primary key autoincrement,
                       name text unique not null,
                       phone text not null,
                       email text,
                       birthdate date);";

      public ContactDatabaseOpenHelper(Context context)
```

```
        {
             super(context, DATABASE_NAME, null, DATABASE_VERSION);
        }
     @Override
     public void onCreate(SQLiteDatabase database)
        {
            database.execSQL(DATABASE_CREATE);
        }
       @Override
       public void onUpgrade(SQLiteDatabase database, int oldVersion, int newVersio
       {
          // We should have backed up the data for restoration after the upgrade
          database.execSQL("DROP TABLE IF EXISTS contacts");
          // Recreate the table
          onCreate(database);
        }
}
```

ContentProvider

```
// Only few key components
public class ContactsContentProvider extends ContentProvider
{
  private ContactDatabaseOpenHelper dbOpenHelper;
@Override
public boolean onCreate()
{
dbOpenHelper = new ContactDatabaseOpenHelper(getContext());
return false;
}
public Cursor query(Uri uri, String[] projection, String selection,
      String[] selectionArgs, String sortOrder)
    {
       // Execute the query and return a cursor
    }
@Override
public Uri insert(Uri uri, ContentValues values)
{
        // ContentValues consists of key-value pairs
        // The key is the column name
        // The value is the value to be inserted in that column
        // Perform an insert query using them
        // Return a URI to the newly created row
    }
@Override
public int update(Uri uri, ContentValues values, String selection,
String[] selectionArgs)
{
        // selection contains the where clause
            // selectionArgs contains the parameters to the where clause
            // Perform the update query and return number of rows updated
        }
    @Override
    public int delete(Uri uri, String selection, String[] selectionArgs)
    {
        // Delete the rows as peer the where clause
            // specified in selection and selectionArgs
            // Return the number of rows
        }
    }
```

An Example of LoaderManager, CursorLoader and SimpleCursorAdapter

```
    public class MainActivity extends Activity implements LoaderCallbacks<Cursor>
    {
    @Override
    protected void onCreate(Bundle savedInstanceState)
    {
    listViewContacts.setVisibility(View.INVISIBLE);
    fillData();
    }
    @Override
    public Loader<Cursor> onCreateLoader(int id, Bundle args)
    {
        String[] projection = ContactDatabaseOpenHelper.allColumns;
        CursorLoader cursorLoader = new CursorLoader(this,
           ContactsContentProvider.CONTENT_URI, projection, null, null,
                                          ContactDatabaseOpenHelper.COLUMN_NAME);
        return cursorLoader;
    }
    @Override
    public void onLoadFinished(Loader<Cursor> loader, Cursor data)
    {
    adapterContacts.swapCursor(data);
    listViewContacts.setVisibility(View.VISIBLE);
    }
    @Override
    public void onLoaderReset(Loader<Cursor> loader)
    {
    adapterContacts.swapCursor(null);
    }
    private void fillData()
    {
        String[] from = new String[] { ContactDatabaseOpenHelper.COLUMN_NAME,
         ContactDatabaseOpenHelper.COLUMN_ID}; // ID is MUST!
        // Fields on the UI to which we map
        int[] to = new int[] { R.id.textViewListViewItem };
            getLoaderManager().initLoader(0, null, this);
        adapterContacts = new SimpleCursorAdapter(this, R.layout.listview_layout,
                                          null, from, to, 0);
        listViewContacts.setAdapter(adapterContacts);
    }
    }
```

Retrieving an Image from a URI

```
InputStream inputStream = context.getContentResolver().openInputStream(contentUri
Bitmap receivedBitmapImage = BitmapFactory.decodeStream(inputStream);
```

Making HTTP GET Request

```
URL url = new URL(urlString);
HttpURLConnection conn =
(HttpURLConnection) url.openConnection();
conn.setReadTimeout(10000 /* milliseconds */);
conn.setConnectTimeout(15000 /* milliseconds */);
conn.setRequestMethod("GET");
conn.setDoInput(true);
conn.connect();
int responseCode = conn.getResponseCode();
if (responseCode == HttpURLConnection.HTTP_OK)
{
is = conn.getInputStream();
// Convert the InputStream into a string
BufferedReader br = null;
br = new BufferedReader(new InputStreamReader(is, "UTF-8"));
StringBuilder sb = new StringBuilder();
String line;
while ((line = br.readLine()) != null)
{
sb.append(line).append('\n');
}
responseString = sb.toString();
}
else
{
responseString = "HTTP GET request for " + urlString +
                 " returned status " + responseCode;
}
```

Making HTTP POST Request

```
HttpURLConnection conn = (HttpURLConnection) url.openConnection();
conn.setReadTimeout(15000);
conn.setConnectTimeout(15000);
conn.setRequestMethod("POST");
conn.setDoInput(true);
conn.setDoOutput(true);
os = conn.getOutputStream();
BufferedWriter writer =
new BufferedWriter(new OutputStreamWriter(os, "UTF-8"));
writer.write(postData);
writer.flush();
writer.close();
os.close();
int responseCode = conn.getResponseCode();
if (responseCode == HttpURLConnection.HTTP_OK)
{
String line;
BufferedReader br = new BufferedReader(new InputStreamReader(
                                   conn.getInputStream()));
while ((line = br.readLine()) != null)
{
responseString += line;
}
}
else
{
responseString = "HTTP POST request for " + urlString +
                 " returned status " +
                 responseCode;
}
```

Using the AsyncTask Class for Performing Asynchronous Operations

```
// Note: AsyncTask is a generic class
// The first type argument (Object in this example) determines the type of
// parameters array passed to doInBackground
// The second type argument, if not Void, is used for displaying progress
// The third type argument indicates the return type of doInBackground
// and the argument type of onPostExecute()
// AsyncTask is an abstract class and must be extended for use
public class AsyncGetRequest extends AsyncTask<Object, Void, String>
{
@Override
protected void onPreExecute()
{
// Called before running the background task
// Runs on the UI thread
// May be used to initiate a progress display
}
@Override
protected String doInBackground(Object... params)
{
// Runs in a background thread
// params is an array
// Must not access the UI
}
protected void onPostExecute(String result)
{
// Called when the background task is over
// result is the return value from doInBackground
// Runs on the UI thread
// May access the UI
}
}
// Invoking the above class
String url = "http://10.0.2.2/androidhttp/getList.php";
// Even for passing one argument, we need an array
Object[] params = new String[] {url};
AsyncGetRequest getRequest = new AsyncGetRequest();
getRequest.execute(params);
```

Opening a File using Implicit Intent

```
     // Open the file by requesting Android to do it
     // Use of implicit
Intent Intent intent = new Intent();
Uri uri = Uri.parse("file://"+directoryPath+"/"+fileName);
intent.setDataAndType(uri, mimeType);
startActivity(intent);
```

Creating a Notification

```
final NotificationManager notificationManager =
               (NotificationManager) getSystemService(NOTIFICATION_SERVICE);
Notification.Builder notificationBuilder = new Notification.Builder(context);
notificationBuilder.setContentTitle("XYZ");
notificationBuilder.setContentText("Missed Call");
notificationBuilder.setSmallIcon(R.drawable.ic_notification_missed_call);
Notification notification = notificationBuilder.build();
// notificationId is an integer id unique for this type of notifications
notificationManager.notify(notificationId, notification);
```

Services

A Service is an application component that can perform long-running operations in the background and does not provide a user interface. Another application component can start a service and it will continue 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 might handle network transactions, play music, perform file I/O, or interact with a content provider, all from the background.

A service may be use in two ways:

A service is "started" when an application component (such as an activity) starts it by calling startService(). Once 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 might download or upload a file over the network. When the operation is done, the service should stop itself.

A service is "bound" when an application component binds to it by calling bindService(). A bound service offers a client-server interface that allows components to interact with the service, send requests, get results, and even do so across process boundries with inter-process 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.

The same 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 which of the callback methods: onStartCommand() to allow components to start it, onBind() to allow binding and both to allow both ways of working.

The Android system will force-stop a service 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, then it is less likely to be killed. If the system kills your service, it restarts it as soon as resources become available again if onStartCommand() returned START_STICKY.

A services runs in the same process as the application in which it is declared and in the main thread of that application, by default. So, if your service performs intensive or blocking operations while the user interacts with an activity from the same application, the service will slow down activity performance. To avoid impacting application performance, you should start a new thread inside the service.

The bindService() method in the invoking activity requires an object of the ServiceConnection class. This object allows us to handle service related events like onServiceConnected and onServiceDisconnected. If we pass the flag Context.BIND_AUTO_CREATE, the service will be automatically created if it is not already running.

Code to Bind to a Service

```
Intent intent = new Intent(context, LocationService.class);
bindService(intent, serviceConnection, Context.BIND_AUTO_CREATE);
```

Code to Unbind from a Service

```
unbindService(serviceConnection);
```

Service Implementation Template

```
public class LocationService extends Service
{
    public void onCreate();
    public int onStartCommand(Intent intent, int flags, int startId);
    public IBinder onBind(Intent intent);
    public boolean onUnbind(Intent intent);
    public void onDestroy();
}
```

Geolocation

Geolocation means finding the current location of the user. This information can be used for geotagging photographs (adding information of the location to a photograph), displaying maps, finding directions, in various location aware applications that display content according to the location of the user, etc.

There are two primary ways of geolocation. A mobile device in an urbun area can usually receive signals from multiple mobile towers and measure their relative signal strength. This can be used along with information about the physical locations of the towers and trigonometry in complicated algorithms to estimate the position of the mobile device. The accuracy of this method is obviously very limited and highly variable. This method is called network-based geolocation and location information thus obtained is called coarse (grained) location. The other method is to use GPS (Global Positioning System). GPS has 24 satellites orbiting the planet. At any time, at any place, a mobile device can receive signals from at least 4 of these satellites if there are no obstructions. This is enough to determine the location of a user with an accuracy of few meters. Location obtained this way is called fine (grained) location. The mobile device must have a GPS receiver, which most smartphones do. The GPS signals are too weak indoors to be useful, which is a major limitation of the system. Another issue is the the mobile receiver taked several minutes to locate the satellites. This problem is solved using A-GPS (Assisted GPS), where the coarse location obtained using the network-based method is used along with known algorithms for finding the approximate location of the satellites at that place at that time. Once the satellites are found, GPS can provide fine-grained location.

The network method incurrs some data charges on the mobile device as it communicates with the mobile towers. The GPS method is completely free as the signals are broadcast by the satellites over a wide area and all devices in the area receive and use them.

Geolocation or geopositioning has many uses; including finding one's location in an unknown place, sending the user's current location to emergency service, mapping, providing directions for travel, location-aware apps, etc. Location-aware apps work by finding the device's current location and delivering results depending on it. For example, location-aware apps may provide information about the nearest ATM, nearby restaurants/hospitals/doctors, movie halls/multiplexes, currently running movies, shopping malls, tourist attractions, etc. A location-aware app can also notify the user when a person in their contact list is in a nearby area. Geolocation is also used for geo-tagging, wherein the location where a photograph is taken is recorded as meta data in the photograph itself.

Checking Whether Network and GPS Providers are Available or Not

```
locationManager = (LocationManager) getSystemService(Context.LOCATION_SERVICE);
isAvailableNetworkProvider = locationManager
                       .isProviderEnabled(LocationManager.NETWORK_PROVIDER);
isAvailableGPSProvider = locationManager
                       .isProviderEnabled(LocationManager.GPS_PROVIDER);
```

Location Listener Class

```
public class MyLocationListener implements LocationListener
{
   @Override
   public void onLocationChanged(final Location loc)
   {
       double latitude = loc.getLatitude();
       double longitude = loc.getLongitude();
       double accuracy = loc.getAccuracy();
       String provider = loc.getProvider();
       Date timeOfLocationFix = new Date(loc.getTime());
       SimpleDateFormat simpleDateFormat = new SimpleDateFormat( "hh:mm:ss.SSSa")
       String timeOfLocationFixString = simpleDateFormat.format(timeOfLocationFix
       Log.i(TAG, "Latitude=" + latitude + " Longitude=" + longitude +
                        " Accuracy=" + accuracy + " Time of fix=" +
                         timeOfLocationFixString + " Provider=" + provider);
   }
   @Override
   public void onProviderDisabled(String provider) { }
   @Override
   public void onProviderEnabled(String provider) { }
   @Override
   public void onStatusChanged(String provider, int status, Bundle extras) { }
}
```

Registering the Location Listner

```
listener = new MyLocationListener();
if (isAvailableNetworkProvider)
        locationManager.requestLocationUpdates(
                  LocationManager.NETWORK_PROVIDER, 4000, 0, listener);
if (isAvailableGPSProvider)
                  locationManager.requestLocationUpdates(
                  LocationManager.GPS_PROVIDER, 4000, 0, listener);
```

Unregistering the Location Listener

```
if (listener != null)
      locationManager.removeUpdates(listener);
```

Audio

In Android, any additional files can be stored in the raw subfolder under resources. The names of the files should be valid for resource ids (all lowercase, no space). All files in the folder are assigned ids like R.raw.filename. Note that the extension will not be part of the id. These ids can be used to access the file in code. The assets directory in Android works in a similar way, but is not a resource directory, hence no ids are assigned to files under the assets directory. We use the MediaPlayer class to play audio.

Audio should be played from a background service.

Android Code to Play an Audio File from the Raw Directory

```
public void playAudio(View view)
{
    Intent objIntent = new Intent(this, PlayAudio.class);
    startService(objIntent);
}
public void stopAudio(View view)
{
      Intent objIntent = new Intent(this, PlayAudio.class);
      stopService(objIntent);
}
public class PlayAudio extends Service
{
    MediaPlayer objPlayer;
public void onCreate()
    {
         super.onCreate();
         objPlayer = MediaPlayer.create(this,R.raw.sleepaway);
    }
    public int onStartCommand(Intent intent, int flags, int startId)
    {
        objPlayer.start();
        if(objPlayer.isLooping() != true)
        {
            Log.d(LOGCAT, "Problem in Playing Audio");
        }
       return 1;
    }
    public void onStop()
    {
        objPlayer.stop();
        objPlayer.release();
    }
    public void onPause()
    {
        objPlayer.stop();
        objPlayer.release();
    }
public void onDestroy()
    {
        objPlayer.stop();
        objPlayer.release();
    }
@Override
    public IBinder onBind(Intent objIndent)
    {
        return null;
    }
}
```

Video

Video can be played in a similar way as audio. We use the VideoView class for playing video.

```
public class MainActivity extends Activity
{
   @Override
   protected void onCreate(Bundle savedInstanceState)
   {
        super.onCreate(savedInstanceState);
        getWindow().setFormat(PixelFormat.TRANSLUCENT);
        VideoView videoHolder = new VideoView(this);
    // if you want the controls to appear
        videoHolder.setMediaController(new MediaController(this));
        Uri video = Uri.parse("android.resource://" + getPackageName()
                        + "/" + R.raw.milesurmeratumhara);
        videoHolder.setVideoURI(video);
        setContentView(videoHolder);
        videoHolder.start();
   }
}
```

Animation

Android supports three types of animations:

- **Property Animation** Introduced in Android 3.0 (API level 11), the property animation system lets you animate properties of any object, including ones that are not rendered to the screen. The system is extensible and lets you animate properties of custom types as well
- **View Animation** View Animation is the older system and can only be used for Views. It is relatively easy to setup and offers enough capabilities to meet many application's needs. There are two types of animations that can be created using this method

- **Tween Animation** Creates an animation by performing a series of transformations on a single image with an Animation Frame Animationcreates an animation by showing a sequence of frames (images) in order with an AnimationDrawable Drawable AnimationDrawable animation involves displaying Drawable resources one after another, like a roll of film. This method of animation is useful if you want to animate things that are easier to represent with Drawable resources, such as a progression of bitmaps.

An Example of Motion Tweening

In motion tweening, an image is taken and a series of operations are performed on it one by one, each for the time duration specified. This can be done in XML or code. Our example uses an XML animation file to define the operations to be performed.

********** activity_main.xml **********

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
android:layout_width="match_parent"
android:layout_height="match_parent"
android:orientation="vertical" >
<TextView android:layout_width="match_parent"
android:layout_height="wrap_content"
android:text="@string/hello_world" />
<ImageView android:id="@+id/ImageView01"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:src="@drawable/android_logo" >
</ImageView>
</LinearLayout>
```

********** res/anim/hyperspace_jump.xml **********

```
<set xmlns:android="http://schemas.android.com/apk/res/android"
                              android:shareInterpolator="false" >
   <scale android:duration="700"
   android:fillAfter="false"
   android:fromXScale="1.0"
   android:fromYScale="1.0"
   android:interpolator="@android:anim/accelerate_decelerate_interpolator"
   android:pivotX="50%"
   android:pivotY="50%"
   android:toXScale="1.4"
   android:toYScale="0.6" />
   <set android:interpolator="@android:anim/accelerate_interpolator"
                              android:startOffset="700" >
         <scale android:duration="400"
          android:fromXScale="1.4"
```

```
            android:fromYScale="0.6"
            android:pivotX="50%"
            android:pivotY="50%"
            android:toXScale="0.0"
            android:toYScale="0.0" />
            <rotate android:duration="400"
             android:fromDegrees="0"
             android:pivotX="50%"
             android:pivotY="50%"
             android:toDegrees="-45"
             android:toYScale="0.0" />
   </set>
</set>
```

********** MainActivity.java **********

```
public class MainActivity extends Activity
{
   @Override
   public void onCreate(Bundle savedInstanceState)
   {
       super.onCreate(savedInstanceState);
       setContentView(R.layout.activity_main);
       ImageView image = (ImageView) findViewById(R.id.ImageView01);
       Animation hyperspaceJump = AnimationUtils.loadAnimation(this, R.anim.hyper
       image.startAnimation(hyperspaceJump);
   }
}
```

The Telephony API

Android provides us almost full access to the system. The biggest example of this is that we may even initiate calls programmatically from our application as illustrated by the following example. We need the android.permission.CALL_PHONE permission for this.

```
Intent callIntent = new Intent(Intent.ACTION_CALL);
callIntent.setData(Uri.parse("tel:0377778888"));
startActivity(callIntent);
```

Sending and Receiving Text Messages (SMS)

We use the class SmsManager for sending and receiving text messages. We need the android.permission.SEND_SMS and android.permission.RECEIVE_SMS permissions respectively. Receiving text messages also requires a broadcast receiver, which must be registered in the manifest. To test the application, create two different AVDs and start both of them at a time. The emulators run on particular TCP ports. The port number is displayed in the title bar of the window. It also doubles up as the telephone number for the emulator. On the receiver side, the message is received in a Bundle as a PDU

(Protocol Data Unit). The method SmsMessage.createFromPdu is used to extract the message from the PDU.

********** AndroidManifest.xml **********

```
...
<uses-permission android:name="android.permission.SEND_SMS" />
<uses-permission android:name="android.permission.RECEIVE_SMS" />
...
<application
   ...
   <receiver android:name=".SMSReceiver" >
       <intent-filter>
       <action android:name="android.provider.Telephony.SMS_RECEIVED" />
       </intent-filter>
   </receiver>
   ...
</application>
```

********** MainActivity.java **********

```
// sends an SMS message to another device
private void sendSMS(String phoneNumber, String message)
{
    SmsManager sms = SmsManager.getDefault();
    sms.sendTextMessage(phoneNumber, null, message, null, null);
}
```

********** SMSReceiver.java **********

```
public class SMSReceiver extends BroadcastReceiver
{
    @Override
    public void onReceive(Context context, Intent intent)
    {
        // ---get the SMS message passed in---
        Bundle bundle = intent.getExtras();
        SmsMessage[] msgs = null;
        String str = "";
        if (bundle != null)
        {
           // ---retrieve the SMS message received---
           Object[] pdus = (Object[]) bundle.get("pdus");
           msgs = new SmsMessage[pdus.length];
           for (int i = 0; i < msgs.length; i++)
           {
                msgs[i] = SmsMessage.createFromPdu((byte[]) pdus[i]);
                str += "SMS from " + msgs[i].getOriginatingAddress();
                str += " :";
                str += msgs[i].getMessageBody().toString();
                str += "\n";
            }
            // ---display the new SMS message---
            Toast.makeText(context, str, Toast.LENGTH_SHORT).show();
        }
    }
}
```

Displaying Web Content

Android provides a WebView class that is capable of displaying web content (web pages). The content is specified either using a URI (if it is to be accessed from the network, android.permission.INTERNET permission would be needed) or even from a string. by default the browser controls (address bar, forward button, back button, etc.), JavaScript, zooming, cookies, etc. are disabled. But they can be enabled by calling appropriate methods. As a WebView is likely to be large in size, usually it is used as the root element of the layout file, but this is not a requirement.

```
WebView webview = new WebView(this);
setContentView(webview);
// Simplest usage: note that an exception will NOT be thrown
// if there is an error loading this page (see below).
webview.loadUrl("http://slashdot.org/");
// OR, you can also load from an HTML string:
String summary = "<html><body>You scored <b>192</b> points.</body></html>";
webview.loadData(summary, "text/html", null);
// ... although note that there are restrictions on what this HTML can do.
```

A WebView has several customization points where you can add your own behavior. These are:

- Creating and setting a **WebChromeClient** subclass. This class is called when something that might impact a browser UI happens, for instance, progress updates and JavaScript alerts are sent here
- Creating and setting a WebViewClient subclass. It will be called when things happen that impact the rendering of the content, eg, errors or form submissions. You can also intercept URL loading here (via shouldOverrideUrlLoading())
- Modifying the **WebSettings**, such as enabling JavaScript with setJavaScriptEnabled() Injecting Java objects into the WebView using the addJavascriptInterface(Object, String) method. This method allows you to inject Java objects into a page's JavaScript context, so that they can be accessed by JavaScript in the page

Capturing Motion Data from Motion Sensors

Mobile devices come with different kinds of motion and position sensore like accelerometer (also known as G-Sensor), gyroscope, compass, etc.

Accelerometers provide data about the acceleration of the device. The change in the position of a thing is called displacement. The rate of chnage of position (including direction change) is called velocity. It is the derivative of the position with respect to time. The rate of change in velocity is called acceleration. Acceleration is the derivative of velocity with respect to time. Accelerometers measure acceleration in one direction. As we live in a three-dimensional space, three accelerometers are needed to measure the change in position in 3-D. Data returned by accelerometer include the acceleration due to the Earth's gravity. The change in position can be calculated by discounting for gravity andthen performing double-integration of acceleration data over time. The results are fairly accurate for short distances, but lose accuracy as the distance increases. In motion-sensor based games, we need to model the movement of various elements accurately as per the laws of physics and data from the accelerometers. This is commonly called physics in gaming terminology and may be done manually for simple games or using ready-made physics engines for more complicated ones.

Accelerometer data can be used in a variety of applications, including rotation of device display (orentation change from portrait mode to lanscape mode and vice versa) when the device is physically rotated, pedometer (step-counting or counting the distance walked), controlling elements in games by device motion, motion-based gestures (like flip the phone to reject a call), etc.

Displaying Accelerometer Data

```
SensorManager sensorManager = (SensorManager) getSystemService(Context.SENSOR_SEF
Sensor sensor = sensorManager.getDefaultSensor(Sensor.TYPE_ACCELEROMETER);
sensorManager.registerListener(new SensorEventListener()
{
   @Override
   public void onSensorChanged(SensorEvent event)
   {
        float x = event.values[0];
        float y = event.values[1];
        float z = event.values[2];
    }
    @Override
    public void onAccuracyChanged(Sensor sensor, int accuracy)
    {
    }
}, sensor, SensorManager.SENSOR_DELAY_FASTEST);
```

Capturing an Image from Camera from within Our App (Using Intent)

Sometimes, we may want to capture an image from within our app. For example, when entering data of a new employee or customer on a mobile device, we may want to capture their photo. Or, while entering data, we may want to capture a photo of some document as a proof. There are some apps that let users sell used things by entering their information the asking price and a photograph. In such cases, we should be able to capture the photograph right from within our app, rather than breaking the flow and asking the user to capture it using a camera app and then selecting the captured image from the gallery.

There are two ways of capturing image from our app. The first, and the simpler, is to invoke an exisitng camera app using an intent. The other is to build our own camera app.

To capture image by invoking some camera app using intent, we need to create an intent with the action

```
MediaStore.ACTION_IMAGE_CAPTURE
```

To capture video by invoking some camera app using intent, we need to create an intent with the action

```
MediaStore.ACTION_VIDEO_CAPTURE
```

Code for capturing an image

```
// App can be installed only of the device has camera
// In manifest file... <uses-feature android:name="android.hardware.camera"
   android:required="true" />
// Check for the existance of camera in code
PackageManager packageManager = context.getPackageManager();
if (packageManager.hasSystemFeature(PackageManager.FEATURE_CAMERA) == false)
{
    Toast.makeText(getActivity(), "This device does not have a camera.",
                                    Toast.LENGTH_SHORT) .show();
    return;
}
// create Intent to take a picture and return control to the
// calling application
Intent intent = new Intent(MediaStore.ACTION_IMAGE_CAPTURE);
// start the image capture Intent
startActivityForResult(intent,CAPTURE_IMAGE_ACTIVITY_REQUEST_CODE);
@Override
protected void onActivityResult(int requestCode, int resultCode, Intent data)
{
    if (requestCode == CAPTURE_IMAGE_ACTIVITY_REQUEST_CODE)
    {
        if (resultCode == RESULT_OK)
        {
            // Image captured
            Bitmap bp = (Bitmap) data.getExtras().get("data");
            imageViewCapturedImage.setImageBitmap(bp);
        }
        else if (resultCode == RESULT_CANCELED)
        {
         // User cancelled the image capture
        } else
        {
          // Image capture failed, advise user
         }
      }
}
```

******