ANDROID

COURSE MATERIAL

NARESH i™
technologies

Opp. Satyam Theatre, Ameerpet, Hyd.
Office: 040- 23746666 Mobile: 9000994008

Index

ANDROID

Android delivers a complete set of software for mobile devices:

- An operating system
- Middleware
- Runtime Environment
- Libraries
- Key mobile applications.

Open

Android was built from the ground-up to enable developers to create compelling mobile applications that take full advantage of all a handset has to offer. It was built to be truly open. For example, an application can call upon any of the phone's core functionality such as making calls, sending text messages, or using the camera, allowing developers to create richer and more cohesive experiences for users. Android is built on the open Linux Kernel. Furthermore, it utilizes a custom virtual machine that was designed to optimize memory and hardware resources in a mobile environment. Android is open source; it can be liberally extended to incorporate new cutting edge technologies as they emerge. The platform will continue to evolve as the developer community works together to build innovative mobile applications.

All applications are created equal

Android does not differentiate between the phone's core applications and third-party applications. They can all be built to have equal access to a phone's capabilities providing users with a broad spectrum of applications and services. With devices built on the Android Platform, users are able to fully tailor the phone to their interests. They can swap out the phone's homescreen, the style of the dialer, or any of the applications. They can even instruct their phones to use their favorite photo viewing application to handle the viewing of all photos.

Breaking down application boundaries

Android breaks down the barriers to building new and innovative applications. For example, a developer can combine information from the web with data on an individual's mobile phone — such as the user's contacts, calendar, or geographic location — to provide a more relevant user experience. With Android, a developer can build an application that enables users to view the location of their friends and be alerted when they are in the vicinity giving them a chance to connect.

Fast & easy application development

Android provides access to a wide range of useful libraries and tools that can be used to build rich applications. For example, Android enables developers to obtain the location of the device, and allows devices to communicate with one another enabling rich peer–to–peer social applications. In addition, Android includes a full set of tools that have been built from the ground

up alongside the platform providing developers with high productivity and deep insight into their applications.

Standard Application Development Vs Mobile Device

Java SDK	Android SDK
Lot of CPU speed-Unlimited memory for allocation	Low speed CPU, very little RAM
We write code, compile, run and debug on the same device(Computer).	Code is written and compile on a PC. But the application runs on a secondary device, the mobile phone. Application, Data are transferred to phone during is done on PC. Emulator is a virtual device phones on PC to speed up development.
End user environment, hardware space can be different. Some bugs may occur on Intel CPU, But not SMF CPUs.	We precisely know the spaces of the target devices. All end users devices have same configuration.
Need not worry about task priorities OS will take care.	Phone functionality is priority #1 on a mobile when there is an incoming call, one application would be closed or paused by the device.
Application crash is not a server issue.	Stability battery usage are very important our application should not hang a phone. If user is listening to music on iPad/iPhone in background and runs our application we must disable sounds in our application.
No middle party between developer and end-user. Users can directly download applications from our websites/DVDs	An app store (Apple iTunes), Android market place are the middle party. Application can be download only through these stores. An approved by app stories a must before we can sell/host applications.
Key board and mouse, large screen size. May or may not have a mic, speaker, web cams…etc. GPS system not available even in lap tops. IPAddress is the only way to gives user's location	Multi touch screen Key board may or may not be available Accelerometer (tilt detection) GPS, Camera, mic and speaker are always there. Our application can access users contact list, call history…etc.

Android is an open source software stack which comes with an operating system, middleware components, key applications like calendar, address book, messaging application and the additional API's which makes mobile application development more flexible and beautiful.

- Android SDK
- Documentation

- No development charges
- No licensing costs
- No deployment fees
- Developers community

Android Platform Overview

Android 4.0

Android 4.0 corresponded to the "IceCreamSandwich" milestone branch, and has an API level of 14.

Android 3.0

Android 2.3 corresponded to the "Honeycomb" milestone branch, and has an API level of 11. In versions 3.0 and was rapidly followed by 3.1 and 3.2 and the API level 12 and 13

Android 2.3

Android 2.3 corresponded to the "Gingerbread" milestone branch, and has an API level of 9. In versions 2.3.3 and higher, the API level is 10.

Android 2.2

Android 2.2 corresponded to the "FroYo" milestone branch, and has an API level of 8.

Android 2.1

Android 2.1 corresponded to the "Eclair" milestone branch, and an API level of 7.

The Eclair branch was also used for 2.0 and 2.0.1; however, both of those releases were quickly obsoleted by the version 2.1 Eclair release. As Android 2.1 includes key bug fixes and improvements not present in 2.0/2.0.1, only Android 2.1 should be used for new devices. As there is no compatibility program for 2.0 or 2.0.1, the officially compatible Eclair-based release is Android 2.1.

Android 1.6

Android 1.6 corresponded to the "Donut" milestone branch, and has an API level of 4.

Android 1.5

Android 1.5 corresponded to the "Cupcake" milestone branch, and has an API level of 3.

Android 1.1

Android 1.1 has an API level of 2. Android 1.1 was known as "Petit Four" internally, though this name was not used officially.

Android 1.0

Was the first release of Android, and has an API level of 1. Since it was the first released version of Android, no platform highlights were prepared for this release.

Versions

1.5 -------> Cupcake
1.6 -------> Donut
2.0/2.1 -----> Eclair
2.2 -------> Froyo
2.3 -------> Gingerbread
3.x -------> HoneyCombo
4.0/4.0.3 ---> Icecream Sandwich
4.1/4.2 ---> Jelly Bean

Platform Version	API Level
Android 4.2	17
Android 4.1	16
Android 4.0.3	15
Android 4.0	14
Android 3.2	13
Android 3.1	12
Android 3.0	11
Android 2.3.3	10
Android 2.3	9
Android 2.2	8
Android 2.1	7
Android 2.0.1	6
Android 2.0	5
Android 1.6	4
Android 1.5	3
Android 1.1	2
Android 1.0	1

List – Android Phones in India

A list of various Android phones which are currently available in India.

Phone Name

Samsung Galaxy I899

Motorola

Motorola Milestone XT720
Motorola Milestone
Motorola Backflip
Motorola Quench XT3
Motorola Charm
Motorola Quench XT5

HTC

HTC Desire
HTC Legend
HTC Magic
HTC Hero
HTC Wildfire
HTC Tattoo

Samsung

Samsung Galaxy S
Samsung Galaxy Spica
Samsung Galaxy i7500
Samsung Galaxy 5
Samsung Galaxy 3

Sony Ericsson

Sony Ericsson Xperia X10
Sony Ericsson Xperia X8
Sony Ericsson Xperia X10 Mini
Xperia X10 Mini Pro

Dell

Dell Streak
Dell XCD35
Dell XCD28

LG

LG Optimus One
LG Optimus GT540
LG GW620

Acer

Acer Liquid
Acer beTouch E110

Videocon
Videocon Zeus V7500

Spice
Spice MI300

Garmin Asus
Garmin Asus A10

Micromax
Micromax Andro A60

Why Android is success?

- The advantage of Android is you develop your own hardware and you will develop your own applications and release into market.
- Android is open source, where as others are proprietary vendors.

Symbian ---------→Nokia
LG
SEMC
Samsung
iOs ----------→ iPhone

Windows M----→ HTC
Samsung
Motorola
HP

BlackBerry------→ BlackBerry

PalmOs ---- ipaq

Android -----→ HTC, LG, Motorola, SonyErricsun, Bell, Celkon, MicroMax, Cisco, Acer, Hp, Tablibs.

Compressed Format

Android	.apk
Symbian	.sis
iPhone	.app
BlackBerry	.cod
J2ME	.jar
Windows	.cab

ANDROID

Android constitute the following components.

- Hardware reference implementation which deals with hardware components like Camera, Bluetooth, WiFi, Display, Flash, Audio, USB, Power Management …etc.

- Linux Kernel integrated with hardware.
- Runtime Environment (DVM)
- Application Framework for designing applications
- User Interface framework
- Libraries.
- Native Applications (pre installed applications).

Hardware reference implementation:-

Android provides a specific hardware reference implementation which provides the capabilities to talk with the software or background services install on the devices.

Linux Kernel:-

It provides basic set of libraries for running the Android application. It acts as a communication channel between the applications and the hardware components.

Runtime Environment:-

Android application run in a separate pre own runtime environment which is called **Dalvik Virtual Machine (DVM).**

Application Framework:-

Application framework provides a way to expose the services which the operating system is providing like Content Providers, databases, Activity Manager, Package Manager, Sensor Manager, Notification Manager, Resources, Location Manager, Telephony Manager, View system .... Etc.

User Interface Framework:-

A UI framework allows the application developers to develop user screens in a more graphical rich way.

Libraries/API:-

It will provide basic libraries and API's which aids in the development process of the application.

Pre-Installed Applications:-

Android stack comes with the set of pre-installed applications like Calendar, Contact book ... etc.

Applications comes with Android Phone

The native applications which come along side with the android operating system are device first.

1. Android market
2. Browser (Webkit engine based browser)
3. E-mail client/applications (Gmail account by default).
4. Widgets/games/calendar ...

5. Media Player
6. Picture Viewer
7. PIM database applications
8. GPS (Global Positioning System) –Google Maps.

Android Market:-

It is a central repository where thousands of android applications can be downloaded at free of cost.

Browser:-

The browser runs on **Webkit** based engine.

E-mail Client:-

Personal e-mail accounts can be configured in a phone using the e-mail client.

Widgets:-

A number of gaming applications and utilities are shipped with the devices.

Media Player:-

A Media Player will be provided by default which supports various audio/video formats like .mp3, .mp4, .wav files.

Picture Viewer:-

A default picture viewer application will come with supports different graphical formats like **.png**, .jpg … (**.png** is preferred)

PIM database:- (Personal Information Management)

The PIM database provides the specialty to store calendar and personal e-mail account.

GPS (Global Positioning System) - Google Maps:-

Androoid provides GPS services for accessing Google Maps.

SMS based Applications:-

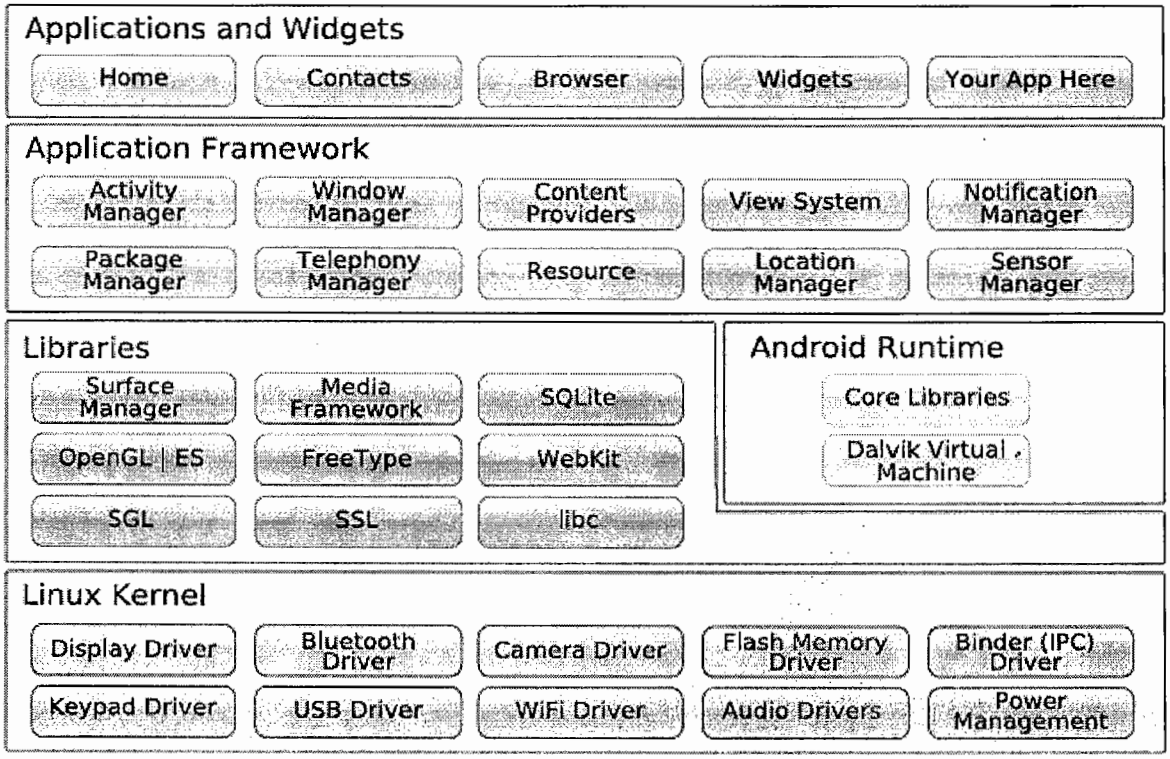
The default SMS applications will allow you to send and receive messages

Features

- **Application framework** enabling reuse and replacement of components
- **Dalvik virtual machine** optimized for mobile devices
- **Integrated browser** based on the open source WebKit engine
- **Optimized graphics** powered by a custom 2D graphics library; 3D graphics based on the OpenGL ES 1.0 specification (hardware acceleration optional)
- **SQLite** for structured data storage
- **Media support** for common audio, video, and still image formats (MPEG4, H.264, MP3, AAC, AMR, JPG, PNG, GIF)
- **GSM Telephony** (hardware dependent)
- **Bluetooth, EDGE, 3G, and WiFi** (hardware dependent)
- **Camera, GPS, compass, and accelerometer** (hardware dependent)
- **Rich development environment** including a device emulator, tools for debugging, memory and performance profiling, and a plugin for the Eclipse IDE

Android Architecture

The following diagram shows the major components of the Android operating system. Each section is described in more detail below.



Applications

Android will ship with a set of core applications including an email client, SMS program, calendar, maps, browser, contacts, and others. All applications are written using the Java programming language.

Application Framework

By providing an open development platform, Android offers developers the ability to build extremely rich and innovative applications. Developers are free to take advantage of the device hardware, access location information, run background services, set alarms, add notifications to the status bar, and much, much more.

Developers have full access to the same framework APIs used by the core applications. The application architecture is designed to simplify the reuse of components; any application can publish its capabilities and any other application may then make use of those .

Underlying all applications is a set of services and systems, including:

- A rich and extensible set of Views that can be used to build an application, including lists, grids, text boxes, buttons, and even an embeddable web browser
- Content Providers that enable applications to access data from other applications (such as Contacts), or to share their own data
- A Resource Manager, providing access to non-code resources such as localized strings, graphics, and layout files
- A Notification Manager that enables all applications to display custom alerts in the status bar
- An Activity Manager that manages the lifecycle of applications and provides a common navigation back stack

Libraries

Android includes a set of C/C++ libraries used by various components of the Android system. These capabilities are exposed to developers through the Android application framework. Some of the core libraries are listed below:

- **System C library** - a BSD-derived implementation of the standard C system library (libc), tuned for embedded Linux-based devices
- **Media Libraries** - based on PacketVideo's OpenCORE; the libraries support playback and recording of many popular audio and video formats, as well as static image files, including MPEG4, H.264, MP3, AAC, AMR, JPG, and PNG
- **Surface Manager** - manages access to the display subsystem and seamlessly composites 2D and 3D graphic layers from multiple applications
- **LibWebCore** - a modern web browser engine which powers both the Android browser and an embeddable web view
- **SGL** - the underlying 2D graphics engine
- **3D libraries** - an implementation based on OpenGL ES 1.0 APIs; the libraries use either hardware 3D acceleration (where available) or the included, highly optimized 3D software rasterizer
- **FreeType** - bitmap and vector font rendering
- **SQLite** - a powerful and lightweight relational database engine available to all applications

Android Runtime

Android includes a set of core libraries that provides most of the functionality available in the core libraries of the Java programming language.

Every Android application runs in its own process, with its own instance of the Dalvik virtual machine. Dalvik has been written so that a device can run multiple VMs efficiently. The Dalvik VM executes files in the Dalvik Executable (.dex) format which is optimized for minimal memory footprint. The VM is register-based, and runs classes compiled by a Java language compiler that have been transformed into the .dex format by the included "dx" tool.

The Dalvik VM relies on the Linux kernel for underlying functionality such as threading and low-level memory management.

Linux Kernel

Android relies on Linux version 2.6 for core system services such as security, memory management, process management, network stack, and driver model. The kernel also acts as an abstraction layer between the hardware and the rest of the software stack.

Android includes the following components. The developer needs to start developing, testing and debugging android applications.

- Android SDK
- Development tools
- Device Emulator
- Documentation
- Sample Source Code
- Online Support

Android SDK:-

The fundamental block of the Android SDK is the Android API/Libraries that provide developer to access the Android software stack which contains different components.

Development Tools:-

Android SDK provides different development tools to convert the written applications into executable programs. Development tools also provide the functionality to test and debug the developed applications.

Android Emulator:-

An Emulator is a fully **interactive skin**, which provides features and functionality of similar to those a real device will be providing. All android emulators run with in the DVM (Dalvik Virtual Machine) making the Android application hardware neutral.

Full Documentation:-

The SDK includes full level reference documentation and code (API) documentation. This makes the development of the application easier and at a faster pace.
Documentation

- Code documentation
- Reference documentation

Code level documentation tells you what is exactly included at what are the packages, each package and each class how you can use them.

Reference level documentation defines how to get started with the basic applications and they give a clear picture of the android fundamentals.

Sample Source Code:-

Android SDK includes feature rich sample applications that demonstrate many possible features android provides you.

Online Support:-

Android provides an excellent developer community using which the developer can build applications in a less time taking the help of the postings/contacts on the developer community.

The most important features which makes android differentiate from other mobile application development platforms are:

- Google Maps and GEO coding
- Background Services
- Data Sharing
- Inter Process Communication
- Peer-to-Peer Inter Device Application Messaging

Google Maps and GEO coding:-

- Google Maps are interaction components which are provided by Google to site the locations on the map.
- Android provides the specialty to integrate Google maps within the application and services we are going to develop.
- Google Map is an automatic and reusable control which you can use in our application.
- The map view widget let the users/developers to display and manipulate a Google map within the activities to build map based application
- Google map applications can be building using the **GoogleMap** interface.

Background Services:-

- Background services provide a specialty to create application that implements an event driven model which works in the background while other applications are running continuously.
- Ex: - A stock market application that tracks the stock prices regularly on a minute-to-minute bases.
- Ex: - Changing the ring tone volume depending on the current location and the identity of caller.

Shared Data:-

- Data can be share between the two different applications using the mechanism of Context Provider.
- When an application creates a database it is totally restricted for use by that application itself.
- The data can be shared between two different applications by providing shared permissions by using Context Provider.

Inter Process Communication:-

- Messages can be exchanged between two different application processor using Intents.

Peer-to-Peer Communication:-

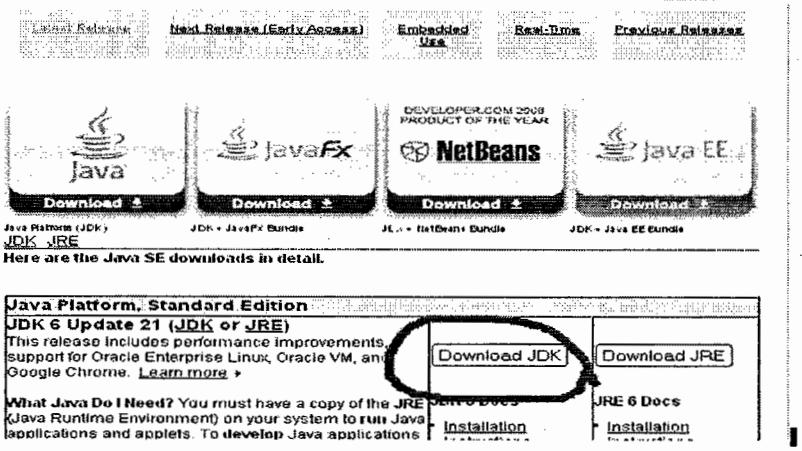
- Android offers peer-to-peer messaging that support instant messaging, present and inter device/inter application communication.
- Inter Application communication enables two application process to exchange messages on behalf of each other.
- Inter device communication enables two applications sitting on different devices to communicate to each other.
- All applications in Android are equal. Android does not differentiate between native applications and third party applications.
- There are two screens which cannot be replaced by the developer.
  1. User Lock Screen
  2. In-call reprieve Screen

Android SDK Installation and Usage.

Please follow the steps mentioned below to install Android SDK and eclipse IDE to get started with application development.

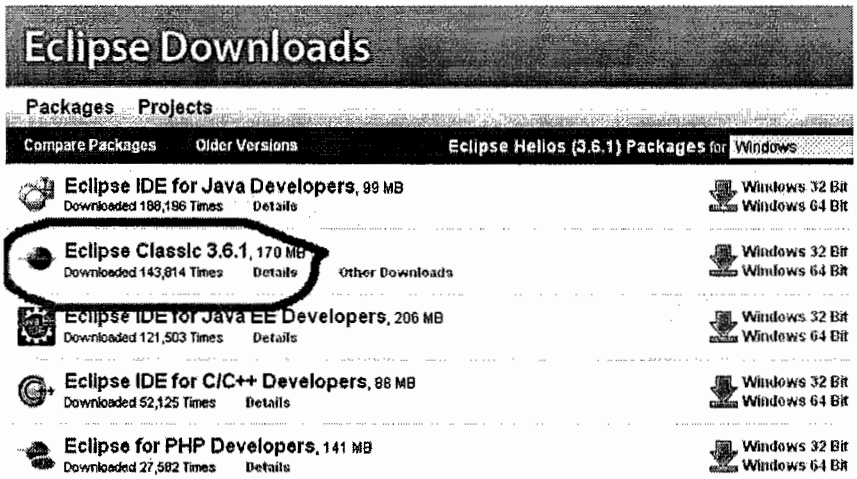
1. Preparing your development machine

Your development system should first download some software before you can program for it. The first one is the JDK which you can download from
http://www.oracle.com/technetwork/java/javase/downloads/index.html



You can download eclipse from here -
http://www.eclipse.org/downloads/

You can use any eclipse IDE above 3.4 but you should always use the latest version.



Note: Make sure you first install the JDK before installing the Eclipse.

Google Recommends
Operating Systems

- Windows XP (32-bit), Vista (32- or 64-bit), or Windows 7 (32- or 64-bit)
- Mac OS X 10.5.8 or later (x86 only)
- Linux (tested on Linux Ubuntu Hardy Heron)
  - 64-bit distributions must be capable of running 32-bit applications.

Supported Development Environments

Eclipse IDE

- Eclipse 3.4 (Ganymede) or 3.5 (Galileo)

Caution: There are known issues with the ADT plug-in running with Eclipse 3.6. Please stay on 3.5 until further notice.

- Eclipse ADT plugin (included in most Eclipse IDE packages)
- If you need to install or update Eclipse, you can download it from http://www.eclipse.org/downloads/.

Several types of Eclipse packages are available for each platform. For developing Android applications, we recommend that you install one of these packages:

- Eclipse IDE for Java EE Developers
- Eclipse IDE for Java Developers

- Eclipse for RCP/Plug-in Developers
- Eclipse Classic (versions 3.5.1 and higher)
- JDK 5 or JDK 6 (JRE alone is not sufficient)
- Android Development Tools plugin (optional)
- Not compatible with Gnu Compiler for Java (gcj)

Other development environments or IDEs

- JDK 5 or JDK 6 (JRE alone is not sufficient)
- Apache Ant 1.6.5 or later for Linux and Mac, 1.7 or later for Windows
- Not compatible with Gnu Compiler for Java (gcj)
- Note: If JDK is already installed on your development computer, please take a moment to make sure that it meets the version requirements listed above. In particular, note that some Linux distributions may include JDK 1.4 or Gnu Compiler for Java, both of which are not supported for Android development.

Hardware requirements

The Android SDK requires disk storage for all of the components that you choose to install. The table below provides a rough idea of the disk-space requirements to expect, based on the components that you plan to use.

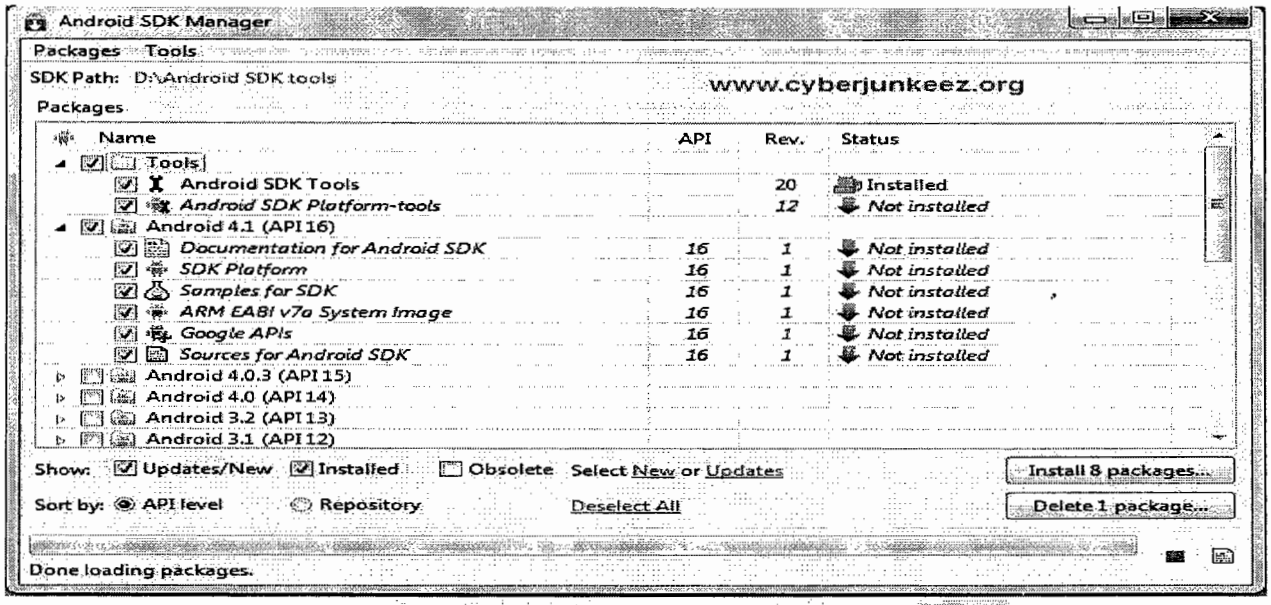
Component type	Approximate size	Comments
SDK Tools	50 MB	Required.
Android platform (each)	150 MB	At least one platform is required.
SDK Add-on (each)	100 MB	Optional.
USB Driver for Windows	10 MB	Optional. For Windows only.
Samples (per platform)	10M	Optional.
Offline documentation	250 MB	Optional.

Note that the disk-space requirements above are in addition to those of the Eclipse IDE, JDK, or other prerequisite tools that you may need to install on your development computer.

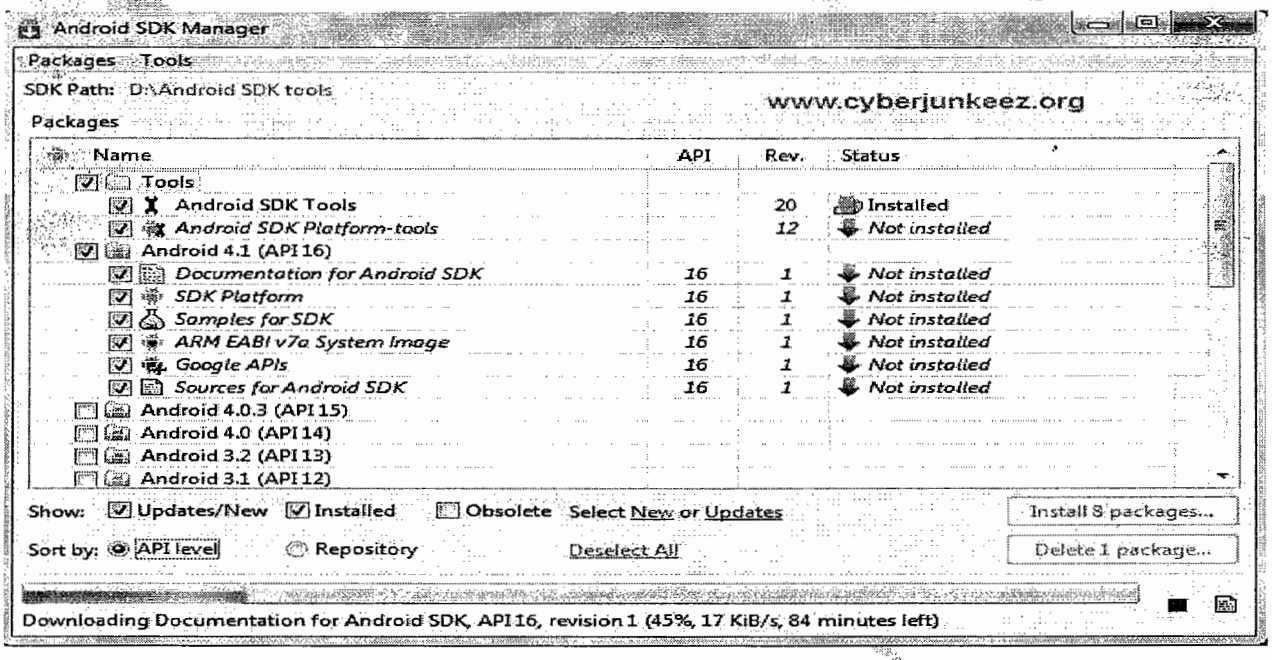
2. How to Install and Run Android 4.1 Jelly Bean SDK on Windows PC?

- Download **Android SDK** from here and Install it on your PC.
- **http://developer.android.com/sdk/index.html**

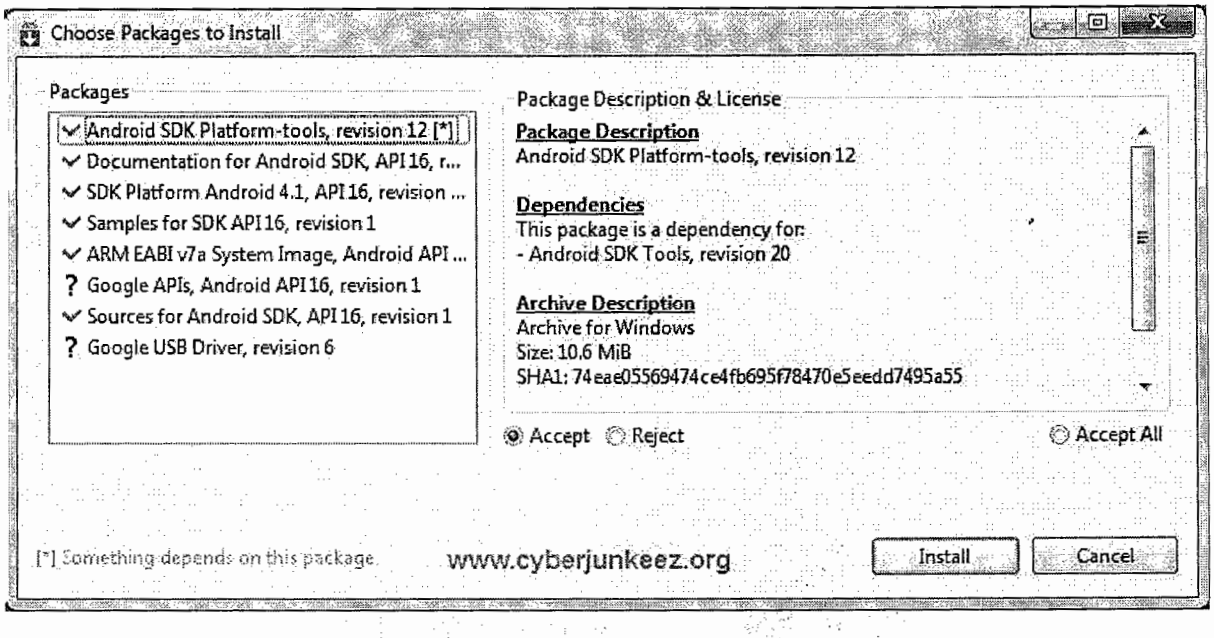
- Open **SDK Manager** from the installed location and wait for the files to get fetched



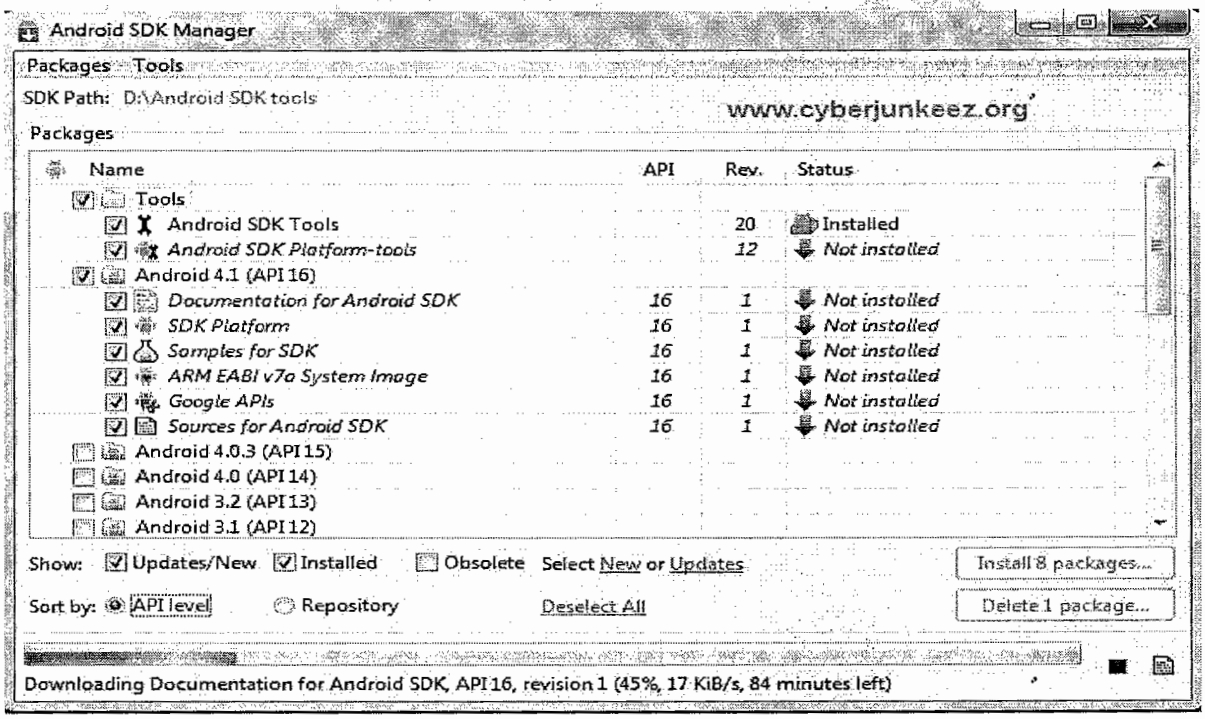
- Once the fetching is complete select the packages you want to install. I selected the 'Tools' and 'Android 4.1 (API 16)' packages. Click Install 8 packages.



- Select Accept and Install.

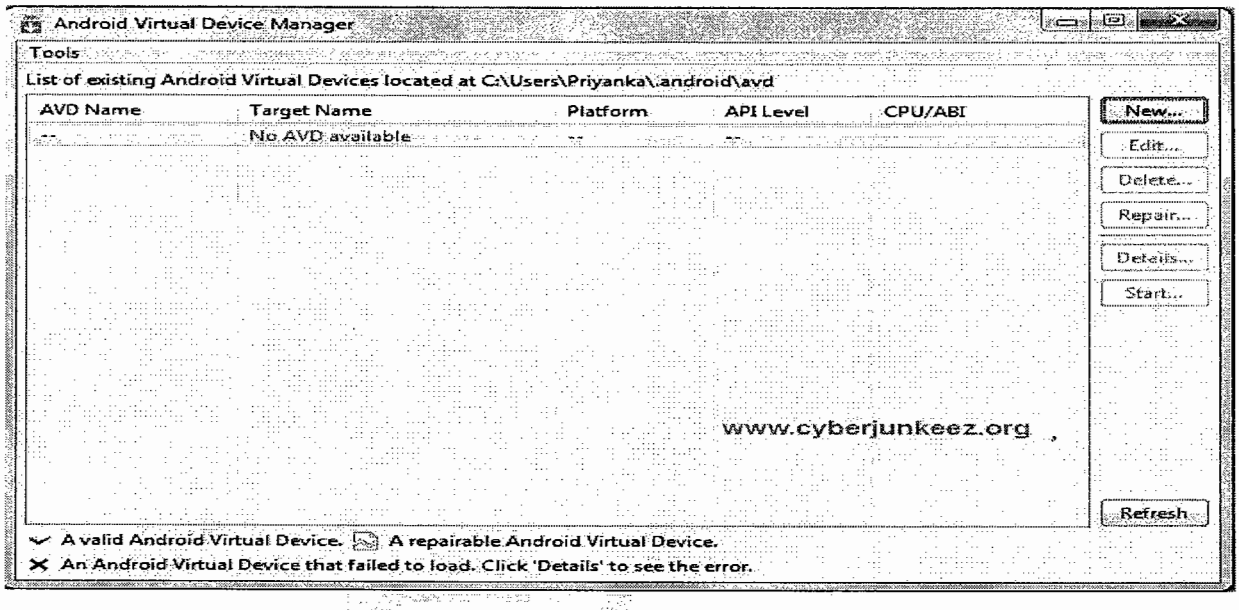


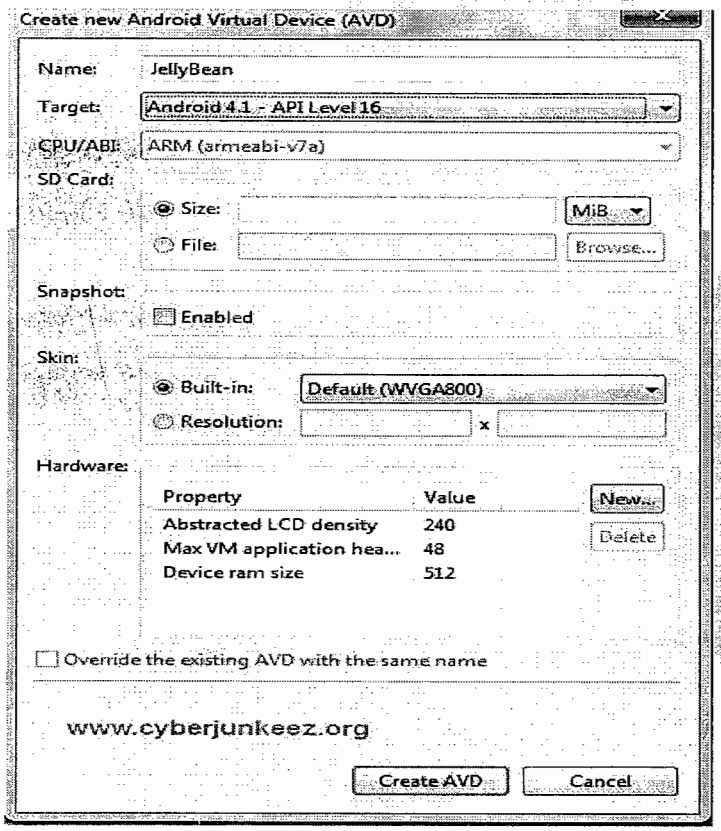
- Wait until the packages are downloaded and installed.

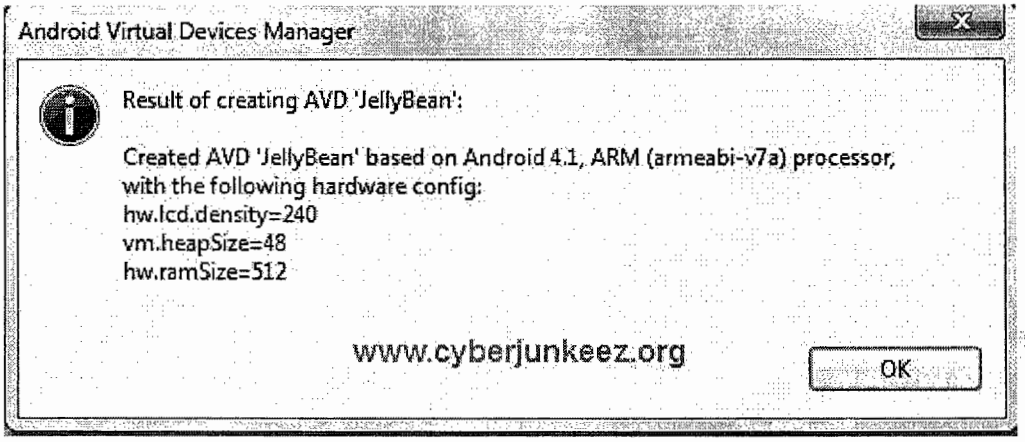


After everything is downloaded, now set up the **Android Emulator** to run Jelly Bean on your PC. Go to Android SDK Tools folder and start **AVD Manager**.

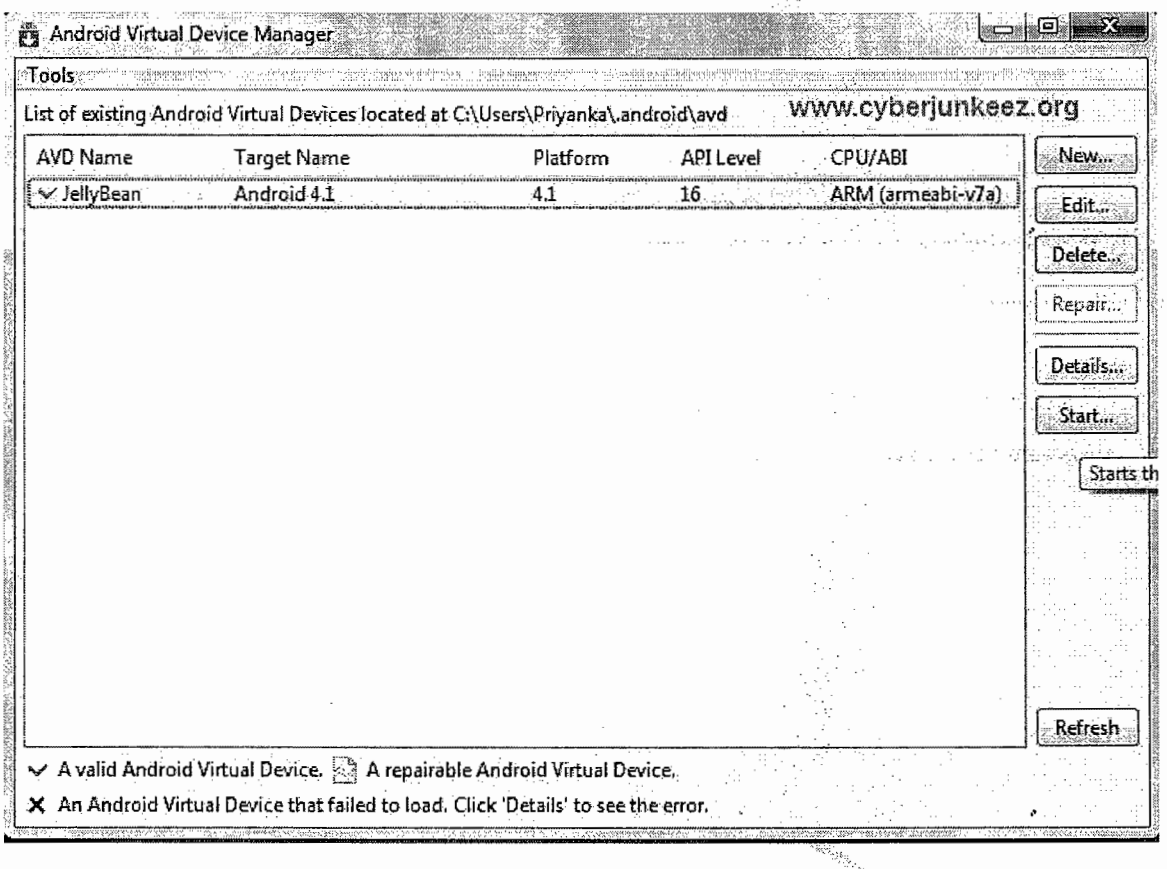
- Click on New and Create a Virtual Device –> Create AVD –> Ok.

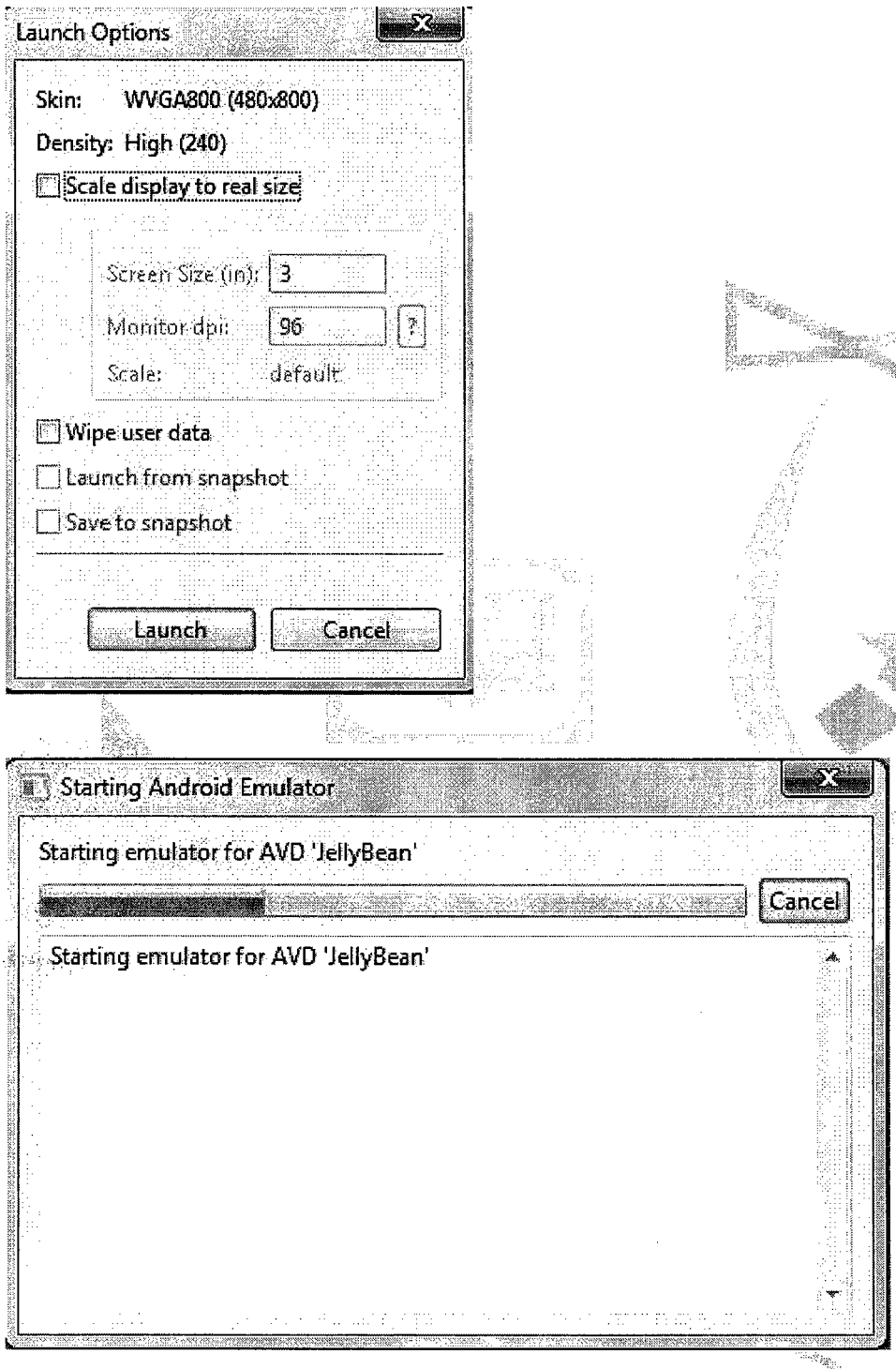

Android Virtual Device Manager
Tools
List of existing Android Virtual Devices located at C:\Users\Priyanka\.android\avd
AVD Name
Target Name
Platform
API Level
CPU/ABI
No AVD available
New...
Edit...
Delete...
Repair...
Details...
Start...
www.cyberjunkeez.org
Refresh
A valid Android Virtual Device.
A repairable Android Virtual Device.
An Android Virtual Device that failed to load. Click 'Details' to see the error.


Create new Android Virtual Device (AVD)
Name:
JellyBean
Target:
Android 4.1 - API Level 16
CPU/ABI:
ARM (armeabi-v7a)
SD Card:
Size:
MiB
File:
Browse...
Snapshot:
Enabled
Skin:
Built-in:
Default (WVGA800)
Resolution:
Hardware:
Property
Value
New...
Abstracted LCD density
240
Max VM application hea...
48
Delete
Device ram size
512
Override the existing AVD with the same name
www.cyberjunkeez.org
Create AVD
Cancel

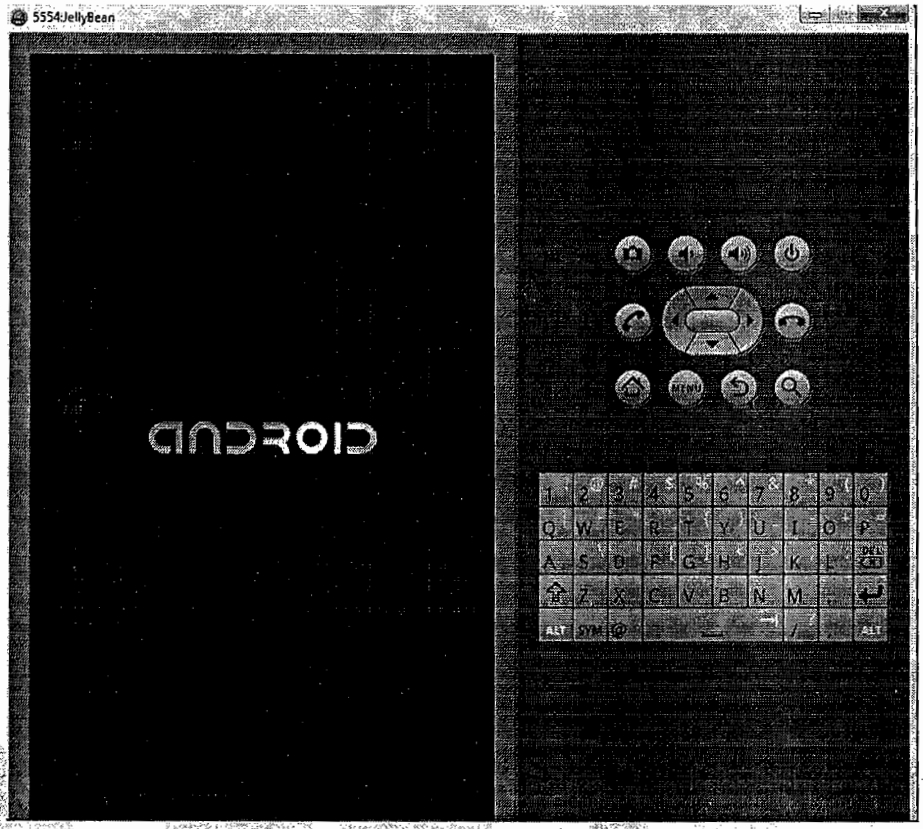


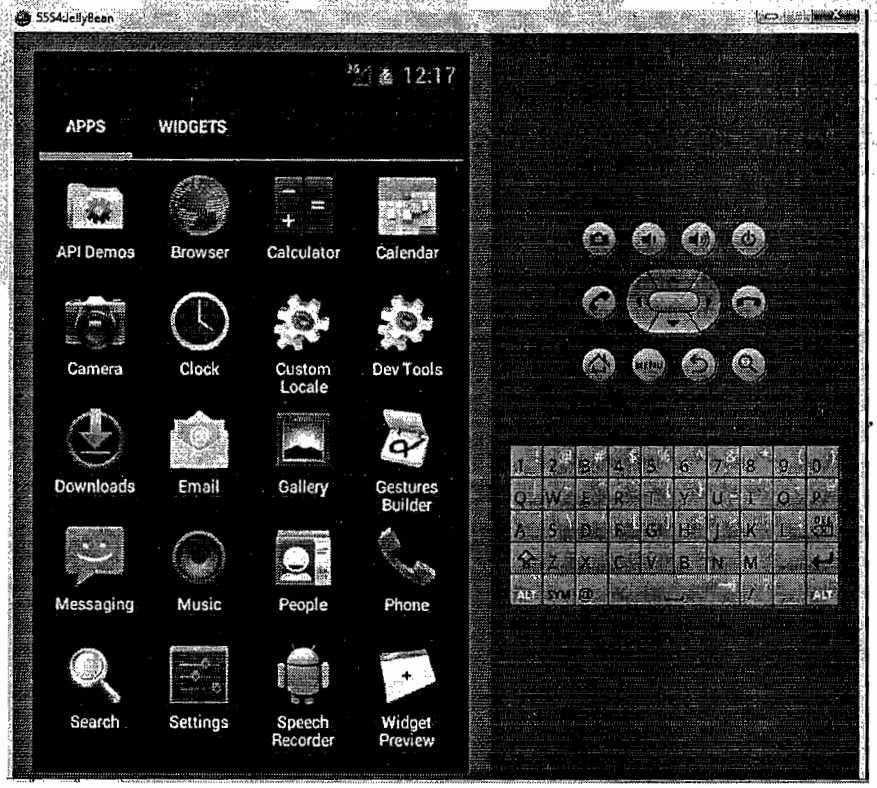
- Now select the virtual device from the list –> Click Start –> Launch.

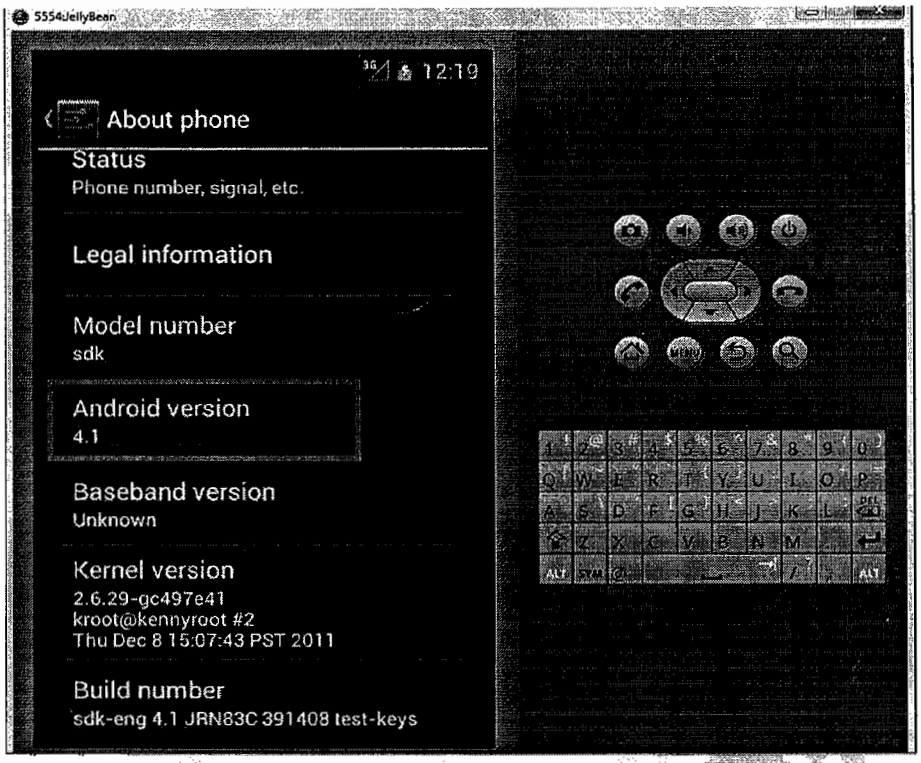




You'll see the following screen after the emulator starts up. The first time boot up will take a few minutes.

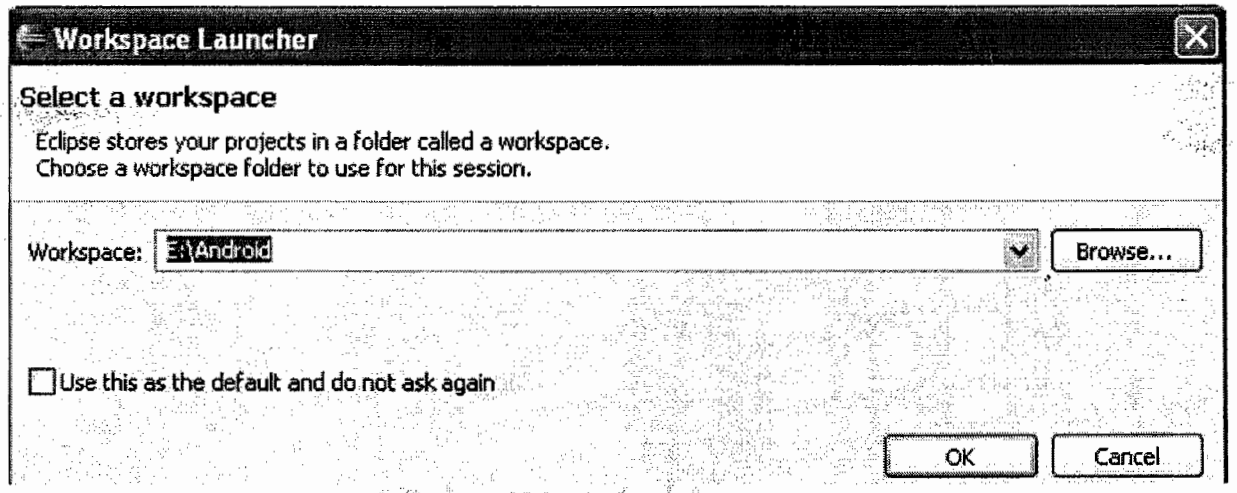

5554:JellyBean
ANDROID


5554:JellyBean
12:17
APPS
WIDGETS
API Demos
Browser
Calculator
Calendar
Camera
Clock
Custom Locale
Dev Tools
Downloads
Email
Gallery
Gestures Builder
Messaging
Music
People
Phone
Search
Settings
Speech Recorder
Widget Preview

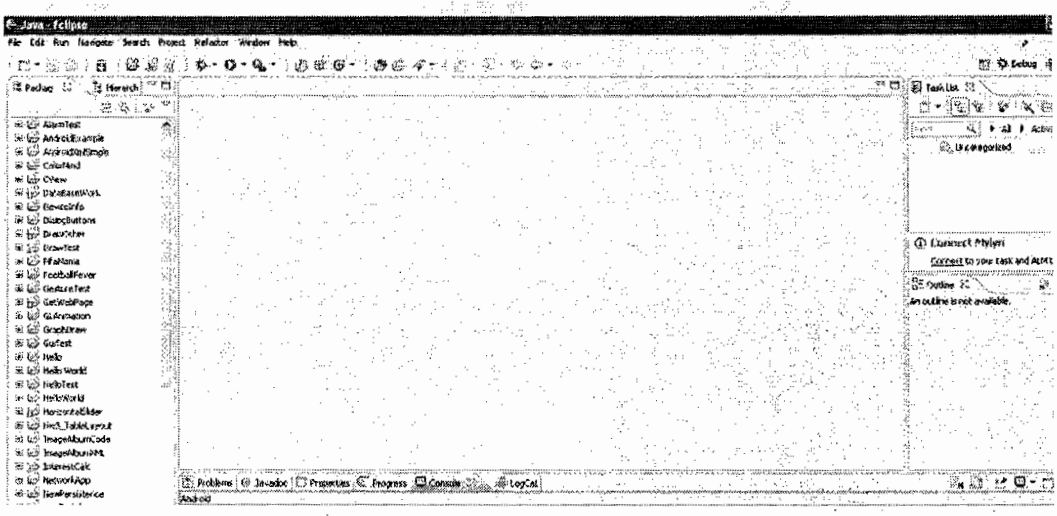


Creating a Hello World App for Android

Now our system is ready for work. Start your eclipse and create a work directory if not already created.



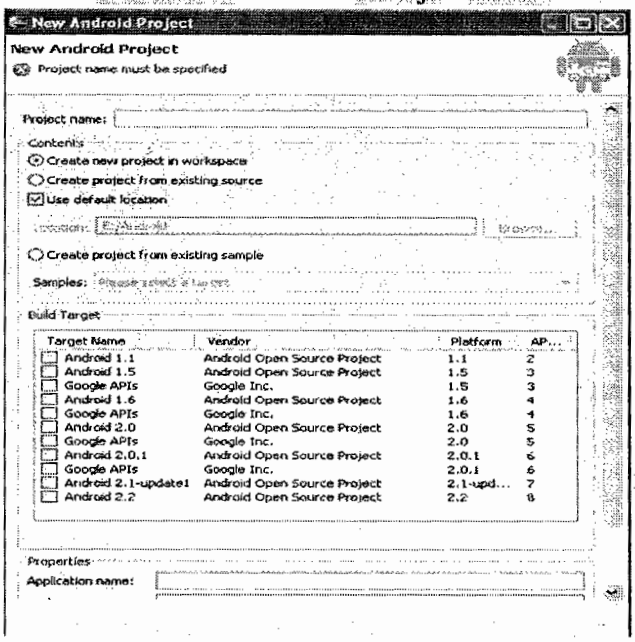
You will see the eclipse ID as shown in the figure below.



The left side of the UI shows the projects created by you. The central UI will show the code written by you and the right end of the figure shows the task lists. The lower end of it shows the console and logger which is very useful while debugging.

Steps to create Hello World Example

1. Open the eclipse IDE in a work space and click
File > New > Other > Android Project and click Next

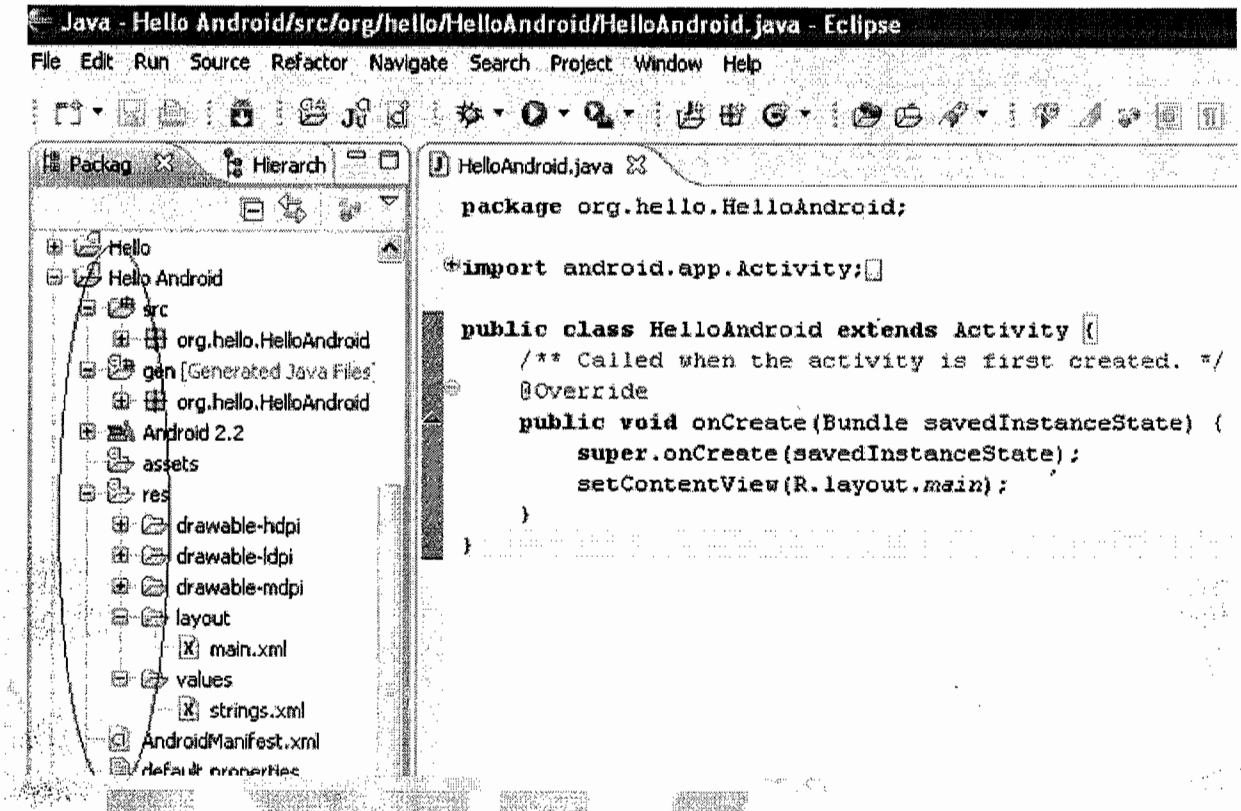


2. Fill the details with project name as "Hello Android". Under contents Select "Create new Project in Workspace" and tick "Use Default Location".
2. Select build target as the latest SDK available. Fill in the properties as follows
Application Name – Hello Android

Package Name – org.hello.HelloAndroid
Create Activity – HelloAndroid

Click "Finish" to create the project in your workspace.

3. You will see the following files created automatically by the SDK.



A new activity java file gets created as shown above contains the onCreate() method which is the first method to be called when the Application starts. Let us look into the file structure. The Master folder is same as Application name it is Hello Android in our case. It contains four subfolders including src, gen, res and Android SDK files.

src – It contains the source packages and java source files. In our src folder it currently contains the package or.hello.HelloAndroid. The package further contains the jave file "HelloAndroid.java".

```
package org.hello.HelloAndroid;

import android.app.Activity;
import android.os.Bundle;
public class HelloAndroid extends Activity {
   /** Called when the activity is first created. */
   @Override
   public void onCreate(Bundle savedInstanceState)
{
```

```
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
    }
}
```

Let us spend some time understanding the code that got auto generated. If you have programmed in java before you can make out most of the codeabove. However if you are new to java and programming, let me clarify few things for you.

- The first line declares the package "org.hello.HelloAndroid". A package is a namespace that organizes a set of related classes and interfaces. So all the folders for this project will be contained in this package and they will contain different elements like images, sound files and java source files.

- The next two lines are importing standard packages for the Android specific java code. Import is the key word which is used to access the standard and no standard packages inside a java file.

- We then create our class **HelloAndroid** which inherits the Activity class. Activity is a standard class of Android which we will discuss in detail a bit later.

- Inside the class we define a method **onCreate()** which is called when the activity is starting. This is where most initialization happens. The **setContentView()** inflates the activity's UI and in our example it is calling the main xml discussed below to draw the user Interface.

- To sum up this class imports standard definitions and create a class which is a subclass of an Activity. The class further has a method called as onCreate() which initializes and paints the UI from a main file. So now that it makes sense lets move ahead.

- **gen** – It contains the auto generated java files. As these files you should make any changes in them. If you make changes in source code the code in this folder will get modified automatically.

- **Android** – It contains the particular SDK libraries being used for the current project.

- **res** – It is one of the other important content folders. It contains three subfolders for images namely drawable –hdpi, drawable –ldpi, drawable –mdpi. The other two subfolders are layout and values. The layout contains the main.xml which is called when the application is started. If you are familiar with c or java programming you know the function main which is called whenever the programs first starts and in a similar way the main.xml draws its content as soon as the application starts. The trings.xml contained in the values folder is used to define strings to be used within the applications.

Let us look into these two important XML files in detail.

main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout  xmlns:android="http://schemas.android.com/apk/res/android"
android:orientation="vertical"
 android:layout_width="fill_parent"
android:layout_height="fill_parent" >
<TextView
android:layout_width="fill_parent"
android:layout_height="wrap_content"
android:text="@string/hello"  />
</LinearLayout>
```

It starts with the LinearLayout tag which implies that we want to put some components on the screen in a linear fashion. There are many other layouts also defined in Android which you will learn later. The orientation, width and height describe how the layout should look. We follow it with a "TextView" component which is used to display texts on the screen. In this example it is taking the text from the "hello" string defined in the "strings.xml".

strings.xml

```
 <?xml version="1.0" encoding="utf-8"?>
<resources>
<string name="hello">Hello World,
HelloAndroid!</string>
<string name="app_name">Hello Android</string>
</resources>
```

From the above content you can easily make out that the hello string corresponds to the actual string "Hello World, HelloAndroid! This is same as the application name we gave.

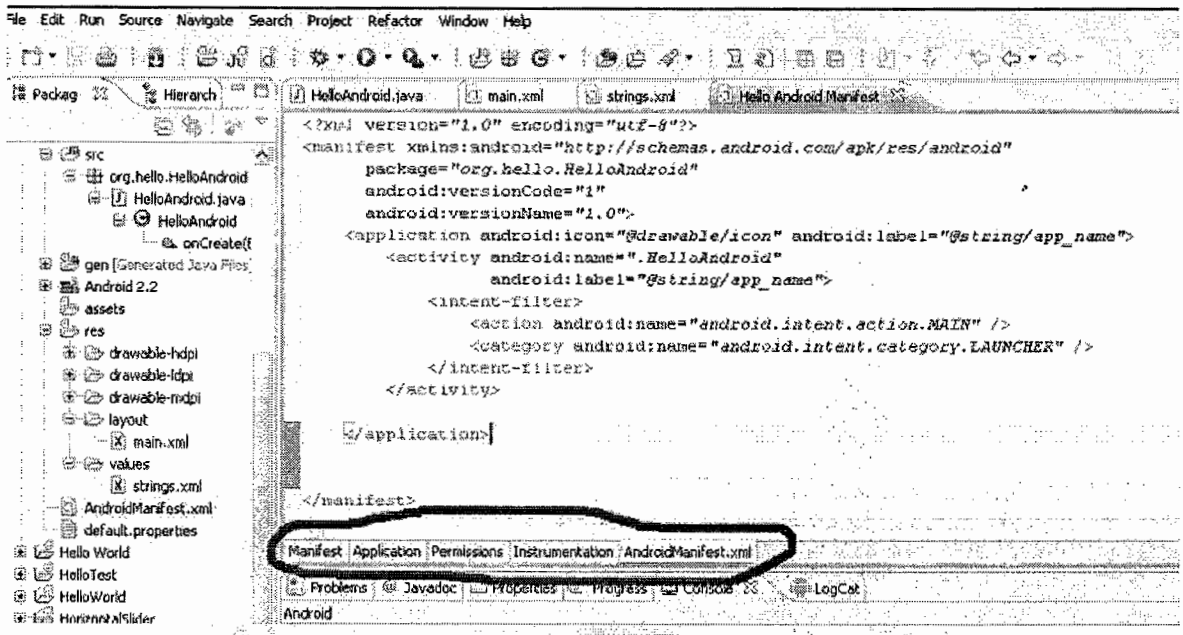
Let us now look into another important xml file created by the SDK for us. It is the android manifest file.

AndroidManifest.xml

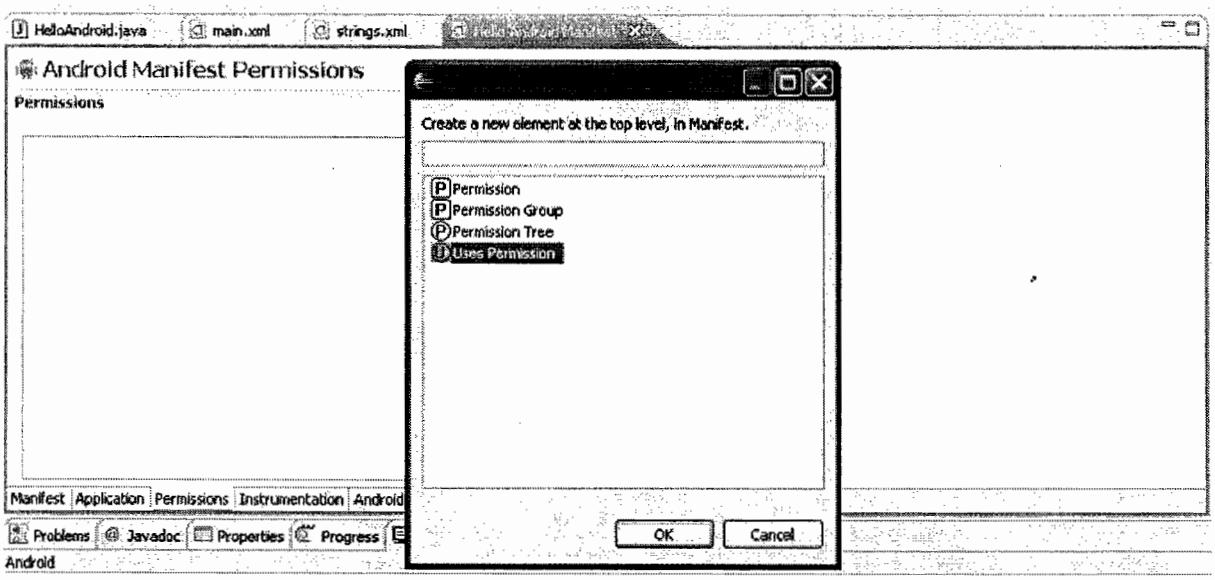
```
 <?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
        package="org.hello.HelloAndroid" android:versionCode="1"
        android:versionName="1.0">
        <application android:icon="@drawable/icon" android:label="@string/app_name">
                <activity android:name=".HelloAndroid"
                                android:label="@string/app_name">
                        <intent-filter>
                                <action android:name="android.intent.action.MAIN" />
                                <category android:name="android.intent.category.LAUNCHER"
                                />
                        </intent-filter>
                </activity>
        </application>
```

</manifest>

This file outlines the main xml and the activity (type of process) which should start after loading the application. You can define specific permissions for the application like Network access and SMS inbox access. The package name, version number and other details about the application is also contained in this file. It primarily consists of four important parts which come together as a XML file. These four components are Manifest, Application, Permission and Instrumentation.

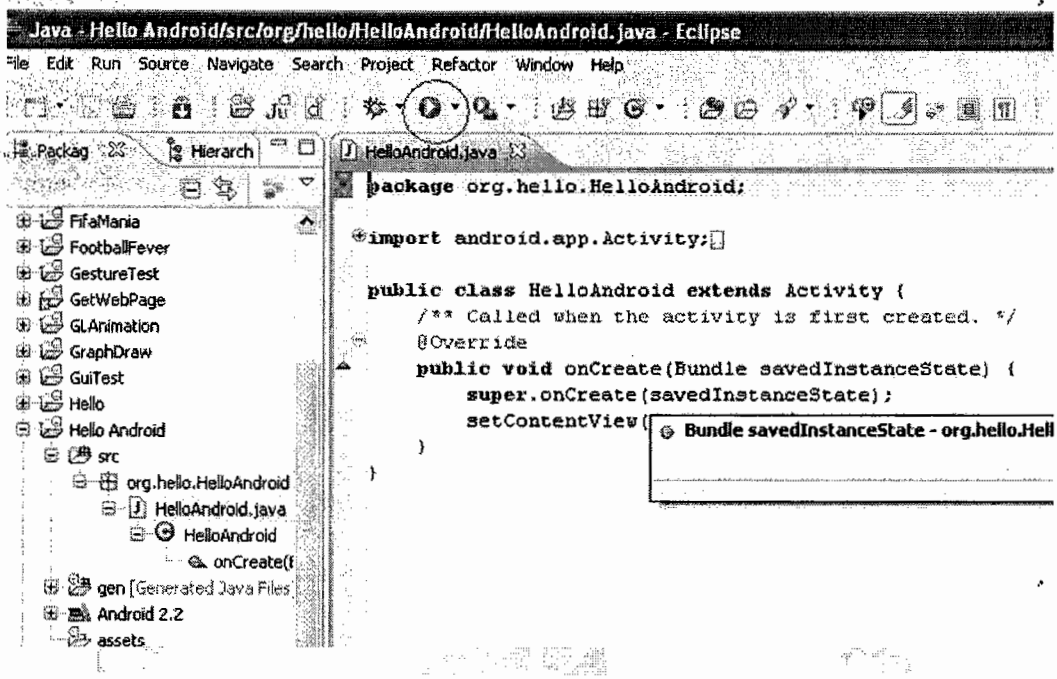


You will be making changes in the four components and the XML will get modified automatically. Like if we are to add permission we will go to the permission tab and add the permission as shown below.

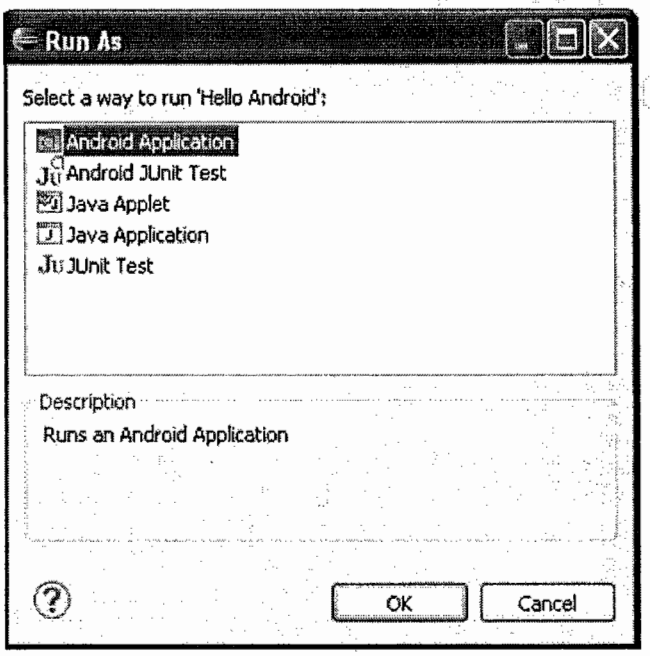


Compiling and executing our Hello World.

It is easy to compile the code and run the device simulator associated with the SDK.



Click on the "Run As" as shown in the figure above. A selection window appears as shown below. Select the Android Application from it.



Once you press ok your code will start compiling and your simulator will start.

Click on the "menu" key on the simulator and you can see the result of the application.

This example of ours was very basic but it still is a complete Android Application and you can take some time to grasp the structure and basic aspects of an Android Application. Let us in the meantime try and modify this program of ours and put some other text in the output screen.

Android Application Life Cycle

- Android run time environment will manage the Resources and the life cycle of the application, android application has no control over their own life cycle. Android application components track the state of the application and they react in a proper manner.
- Android managed resources in such a way that the application reminds responsive for the end users. The management of the resources is based on prioritization process.

Application priority-

The priority of the hosted applications determines the process in which, the processes are killed and the resources are taken back. The priority of an application is equal to the highest priority component.

When two applications are having the same priority the process, that was staying for a longer time will be killed first.

In the case of inter process communication if an application is dependent on the service offered by other application, the other application will have the same priority as the first application which is offering the service.

States of the process:-

1. Active process (Highest priority Ex:- Gallery)
2. Visible process (High second priority Ex:- Dialog Box)
3. Started service process (High second priority Ex:- Video started but not visible)
4. Background process (Low priority Ex:- Media player)
5. Empty process (Low priority Ex :- Second time accessing application)--->The old instance is stores in the empty process.

Active process:-

- Active process or Foreground process is the application with components currently interacting with the user. This process will be running in a few number and they will be killed at the last priority. Active process or Service or Background process which are currently executing receiving event handlers.
- Services that are executing onStart(),onCreate() and onDestroy() event handlers also come under Active process.

Visible process:-

- These are the processes that will be running visible activities.
- It is the activities which will not be responsible to user events.

Ex: - After call end we get the balance amount…

Started service process:-

- These are the processes which will be hosted services that have been started.

These processes are considered to be foreground process and they will not be killed unless resources are needed for active or visible process.

Background process:-

- These are the processes which are not visible and which don't have services running in the background. Android runtime management will kill the process using a last-seen-first-killed (lsfk) pattern or LRU (Last recently used).

Empty process:-

- Android defines the applications in the memory even after they reach, they end of the life time. Android maintains this to improve the startup time to the applications. When they are re-launched/re-started. These by the android runtime environment whenever required.

Life Cycle of an Activity

Visible life time
onStart() to onStop().

Foreground life time
onResume() to onPause().

Entire life time
onCreate() to onDestroy().

An Activity comprises of 3 states.

1. **Active or Running state:-** when the activity is in the foreground display on screen. The activities will also interacting with the user in this state.
2. **Paused State:-**
   The Activity will be still visible to the user but the user will not be interacting with the activity. Another Activity will be present on top of this activity.

   A paused Activity is completely alive this is maintains the state activity and all other information. But it can be killed by the system in the case of e4xtremely low memory accusations.
3. **Stopped State:-** An activity is said to be in the stopped state when it is completely discarded by another activity, this activity is no longer visible to the user and is often killed by the manager/system when memory us needed on other activities or applications.

Entire Life Time

The entire life time of the activities of the activity access between the call to onCreate() through a final call to onDestroy() methods.

An activity performs all its startup tasks on the call to onCreate() method. It releases all the resources back to the system through the call to onDestroy() method.

Visible Life Time

The activity occurs between a call to onStart() through the call to onStop() method. During this life time the activity will not be interacting with the user. Between these two activities resources are maintained that are needed by the system to show the activity on the screen.

Foreground Life Time

The activity occurs between a call to onResume() through a call to onPause() method. During this life time the activity will be in front of all other activities on the screen and will be interacting with the end user.

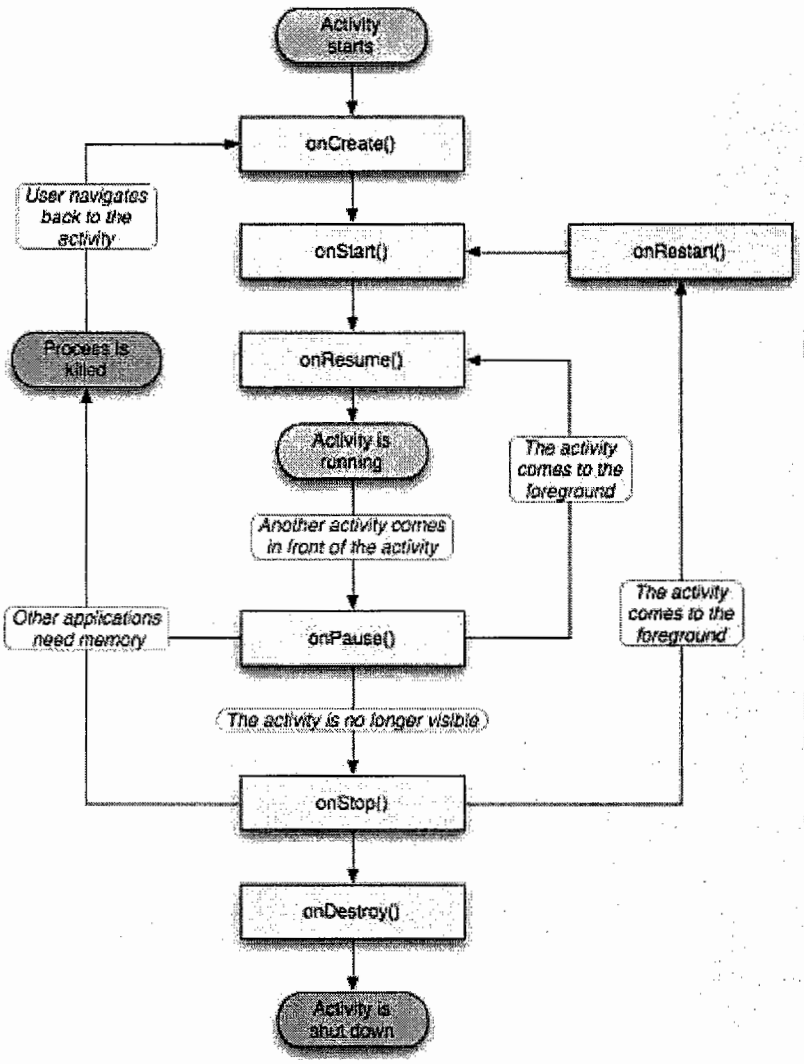
Activity Lifecycle Methods.

onCreate():-

This method is called when the activity is first built. At this step all the startup activities are performed that are required by the activity in the for other steps.

onStart() :-

This method is called just before the activity becomes visible to the user. This method is followed by onResume() method. If the activity comes to the foreground or onStop() method, if it is going into a hidden state.



onResume() :-
This method is called before the activity starts interacting with the user. This method is always followed by onPause() method.

onPause() :-
This method is called by the activity when the system is about to start on other activity.

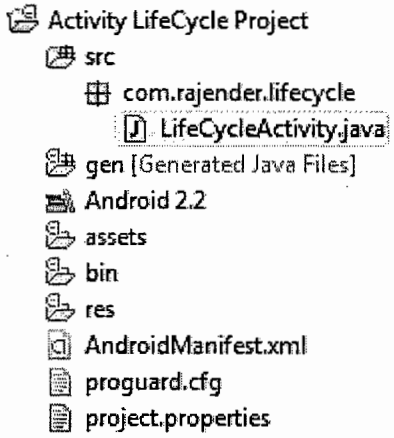
This method is used to commit un saved changes, stop all the graphical user interfaces which involves heavy weight components or any other item that consumes the processing power with this followed by onResume() if the activity returns to the foreground or onStop(), if it becomes invisible to the user.

onStop() :-
This method is called when the activity is no longer visible to the end user. This occurs became on other activity has been resumed or because it is being destroyed.

onDestroy() :-
It is called before the activities destroyed. The activity receives this method as the final call before it ends.

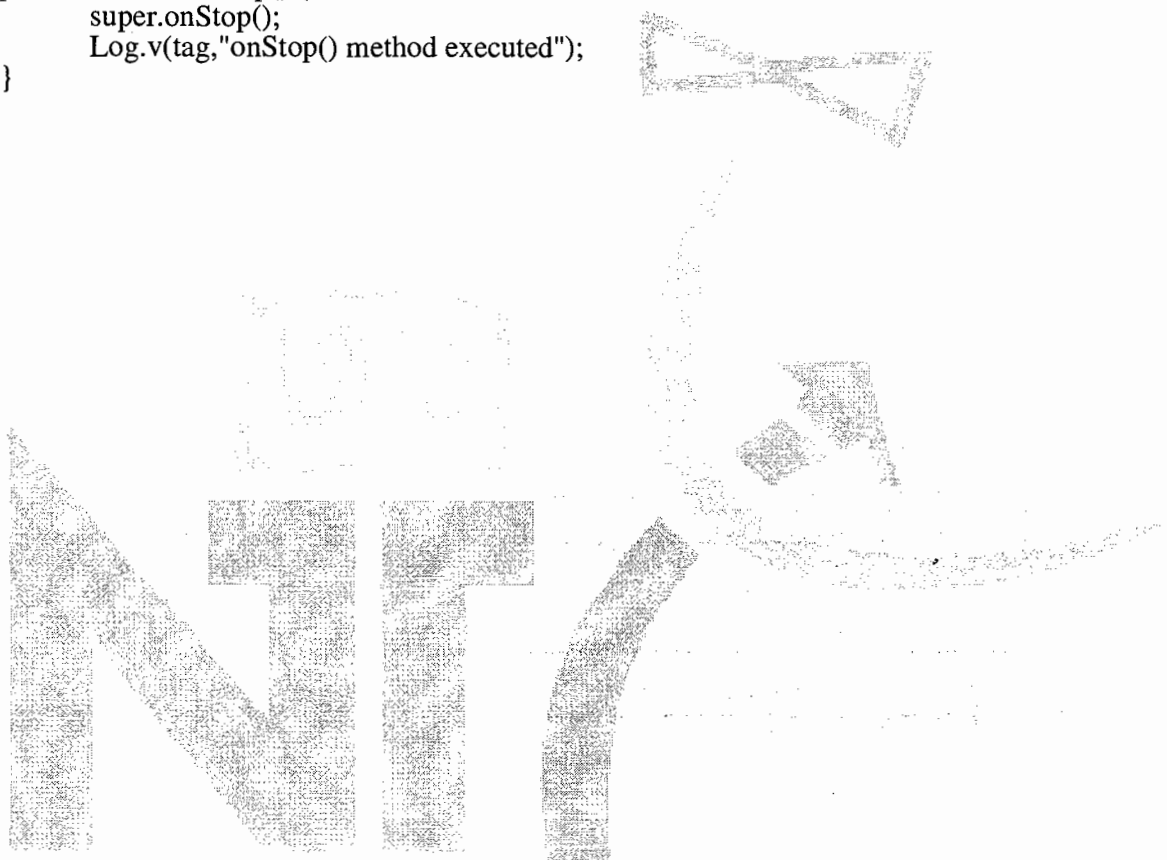
onRestart() :-
This is called after the activity has been stopped and before it is getting started again.

Activity Life Cycle

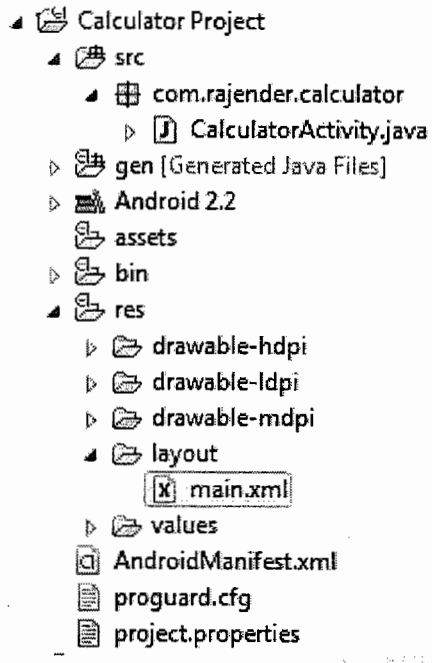



```
package com.rajender.lifecycle;

import android.app.Activity;
import android.os.Bundle;
import android.util.Log;
public class LifeCycleActivity extends Activity {
   public static final String tag="LifeCycleActivity";
        @Override
        public void onCreate(Bundle b) {
                super.onCreate(b);
                setContentView(R.layout.main);
                Log.v(tag,"onCreate() method executed");
        }
        @Override
        protected void onDestroy() {
                super.onDestroy();
                Log.v(tag,"onDestroy() method executed");
        }
        @Override
        protected void onPause() {
                super.onPause();
                Log.v(tag,"onPause() method executed");
        }
        @Override
        protected void onRestart() {
                super.onRestart();
                Log.v(tag,"onRestart() method executed");
        }
        @Override
        protected void onResume() {
                super.onResume();
```

```
        Log.v(tag,"onResume() method executed");
    }
    @Override
    protected void onStart() {
        super.onStart();
        Log.v(tag,"onStart() method executed");
    }
    @Override
    protected void onStop() {
        super.onStop();
        Log.v(tag,"onStop() method executed");
    }
}
```



Basic User Interface Components



CalculatorActivity.java

```
package com.rajender.calculator;

import android.app.Activity;
import android.os.Bundle;
import android.view.View;
import android.widget.EditText;

public class CalculatorActivity extends Activity{
   EditText firstET, secondET, result;
   int a,b,c;
        @Override
   public void onCreate(Bundle b) {
      super.onCreate(b);
      setContentView(R.layout.main);
      firstET=(EditText)findViewById(R.id.firstText);
      secondET=(EditText)findViewById(R.id.secondText);
      result=(EditText)findViewById (R.id.resultText);

   }

        public void calcFunction(View v) {
                String str1 = firstET.getText()
                                                .toString().trim();
                String str2 = secondET.getText()
```

```
                                                .toString().trim();
            a=Integer.parseInt(str1);
            b=Integer.parseInt(str2);

            switch (v.getId()) {
            case R.id.addButton:
                    c=a+b;
                    result.setText(""+c);
                    break;

            case R.id.subButton:
                    c=a-b;
                    result.setText(""+c);
                    break;

            case R.id.prodButton:
                    c=a*b;
                    result.setText(""+c);
                    break;

            case R.id.divButton:
                    c=a/b;
                    result.setText(""+c);
                    break;
                }
        }
}
```

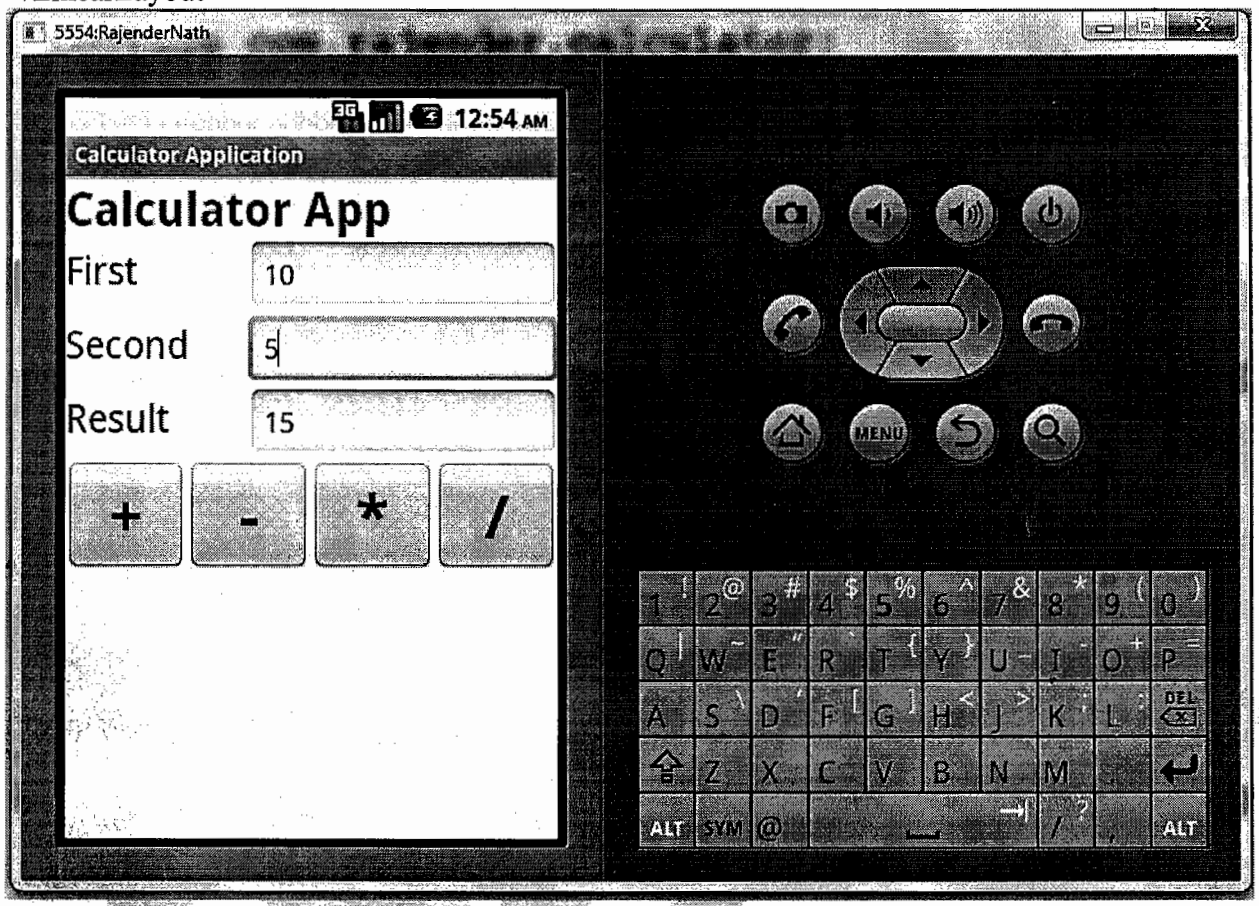
main.xml

```
<?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"
   android:background="#FFFACD"
    >

   <TextView
      android:layout_width="fill_parent"
      android:layout_height="wrap_content"
      android:text="Calculator App"
      android:textSize="30px"
      android:textStyle="bold"
      android:textColor="#f00" />
```

```
<TableRow
   android:layout_width="match_parent"
   android:layout_height="wrap_content" >

   <TextView
      android:layout_width="120px"
      android:layout_height="wrap_content"
      android:text="First"
      android:textColor="#00f"
      android:textSize="25px"/>

   <EditText
      android:id="@+id/firstText"
      android:layout_width="fill_parent"
      android:layout_height="wrap_content" />
</TableRow>

  <TableRow
   android:layout_width="match_parent"
   android:layout_height="wrap_content" >

   <TextView
      android:layout_width="120px"
      android:layout_height="wrap_content"
      android:text="Second"
      android:textColor="#00f"
      android:textSize="25px"/>

   <EditText
      android:id="@+id/secondText"
      android:layout_width="fill_parent"
      android:layout_height="wrap_content" />
</TableRow>
<TableRow
   android:layout_width="match_parent"
   android:layout_height="wrap_content" >

   <TextView
      android:layout_width="120px"
      android:layout_height="wrap_content"
      android:text="Result"
      android:textColor="#00f"
      android:textSize="25px"/>

   <EditText
```

```
        android:id="@+id/resultText"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content" />
    </TableRow>
    <TableRow
      android:layout_width="match_parent"
      android:layout_height="wrap_content" >

      <Button
        android:id="@+id/addButton"
        android:onClick="calcFunction"
        android:text="+"
        android:textSize="40px"
        android:textStyle="bold"
        android:textColor="#f00"
        android:layout_width="80px"
        android:layout_height="wrap_content"
        />
      <Button
        android:id="@+id/subButton"
        android:onClick="calcFunction"
        android:text="-"
        android:textSize="40px"
        android:textStyle="bold"
        android:textColor="#f00"
        android:layout_width="80px"
        android:layout_height="wrap_content"
        />
      <Button
        android:id="@+id/prodButton"
        android:onClick="calcFunction"
        android:text="*"
        android:textSize="40px"
        android:textStyle="bold"
        android:textColor="#f00"
        android:layout_width="80px"
        android:layout_height="wrap_content"
        />
      <Button
        android:id="@+id/divButton"
        android:onClick="calcFunction"
        android:text="/"
        android:textSize="40px"
        android:textStyle="bold"
        android:textColor="#f00"
        android:layout_width="80px"
```

```
        android:layout_height="wrap_content"
        />
    </TableRow>
</LinearLayout>
```



Application Fundamentals

1. Application Components
    1. Activating components: intents
    2. Shutting down components
    3. The manifest file
    4. Intent filters

Key classes

1. Activity
2. Service
3. BroadcastReceiver
4. ContentProvider

5. Intent

Android applications are written in the Java programming language. The compiled Java code — along with any data and resource files required by the application — is bundled by the aapt tool into an *Android package*, an archive file marked by an ***.apk*** suffix. This file is the vehicle for distributing the application and installing it on mobile devices, these file users download to their devices. All the code in a single .apk file is considered to be one *application*.

In many ways, each Android application lives in its own world:

- By default, every application runs in its own Linux process. Android starts the process when any of the application's code needs to be executed, and shuts down the process when it's no longer needed and system resources are required by other applications.
- Each process has its own virtual machine (VM), so application code runs in isolation from the code of all other applications.
- By default, each application is assigned a unique Linux user ID. Permissions are set so that the application's files are visible only to that user and only to the application itself — although there are ways to export them to other applications as well.

It's possible to arrange for two applications to share the same user ID, in which case they will be able to see each other's files. To conserve system resources, applications with the same ID can also arrange to run in the same Linux process, sharing the same VM.

Application Components

A central feature of Android is that one application can make use of elements of other applications (provided those applications permit it). For example, if your application needs to display a scrolling list of images and another application has developed a suitable scroller and made it available to others, you can call upon that scroller to do the work, rather than develop your own. Your application doesn't incorporate the code of the other application or link to it. Rather, it simply starts up that piece of the other application when the need arises.

For this to work, the system must be able to start an application process when any part of it is needed, and instantiate the Java objects for that part. Therefore, unlike applications on most other systems, Android applications don't have a single entry point for everything in the application (no main() function, for example). Rather, they have essential *components* that the system can instantiate and run as needed. There are four types of components:

Activities

An *activity* presents a visual user interface for one focused endeavor the user can undertake. For example, an activity might present a list of menu items users can choose from or it might display photographs along with their captions. A text messaging application might have one activity that shows a list of contacts to send messages to, a second activity to write the message to the chosen contact, and other activities to review old messages or change settings. Though they work together to form a cohesive user

interface, each activity is independent of the others. Each one is implemented as a subclass of the Activity base class.

An application might consist of just one activity or, like the text messaging application just mentioned, it may contain several. What the activities are, and how many there are depends, of course, on the application and its design. Typically, one of the activities is marked as the first one that should be presented to the user when the application is launched. Moving from one activity to another is accomplished by having the current activity start the next one.

Each activity is given a default window to draw in. Typically, the window fills the screen, but it might be smaller than the screen and float on top of other windows. An activity can also make use of additional windows — for example, a pop-up dialog that calls for a user response in the midst of the activity, or a window that presents users with vital information when they select a particular item on-screen.

The visual content of the window is provided by a hierarchy of views — objects derived from the base View class. Each view controls a particular rectangular space within the window. Parent views contain and organize the layout of their children. Leaf views (those at the bottom of the hierarchy) draw in the rectangles they control and respond to user actions directed at that space. Thus, views are where the activity's interaction with the user takes place. For example, a view might display a small image and initiate an action when the user taps that image. Android has a number of ready-made views that you can use — including buttons, text fields, scroll bars, menu items, check boxes, and more.

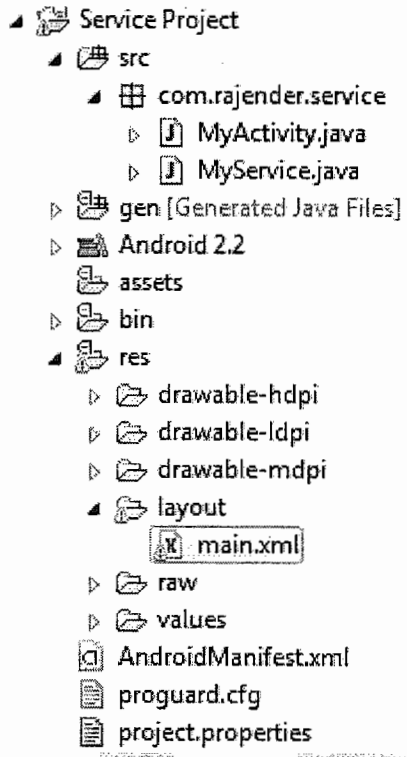
A view hierarchy is placed within an activity's window by the Activity.setContentView() method. The *content view* is the View object at the root of the hierarchy.

Services

A *service* doesn't have a visual user interface, but rather runs in the background for an indefinite period of time. For example, a service might play background music as the user attends to other matters, or it might fetch data over the network or calculate something and provide the result to activities that need it. Each service extends the Service base class.

A prime example is a media player playing songs from a play list. The player application would probably have one or more activities that allow the user to choose songs and start playing them. However, the music playback itself would not be handled by an activity because users will expect the music to keep playing even after they leave the player and begin something different. To keep the music going, the media player activity could start a service to run in the background. The system would then keep the music playback service running even after the activity that started it leaves the screen.

Service Example



main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
   android:layout_width="fill_parent"
   android:layout_height="fill_parent"
   android:orientation="vertical"
   >

   <TextView
      android:layout_gravity="center_horizontal"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="Audio Application"
      android:textSize="30px" />

   <Button
      android:onClick="startFunction"
```

```
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="Start" />

  <Button
    android:onClick="stopFunction"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="Stop" />

</LinearLayout>
```

MyActivity.java

```
package com.rajender.service;

import android.app.Activity;
import android.content.Intent;
import android.os.Bundle;
import android.view.View;

public class MyActivity extends Activity {
  Intent serviceIn;
      @Override
  public void onCreate(Bundle b) {
    super.onCreate(b);
    setContentView(R.layout.main);

    serviceIn=new Intent(this,MyService.class);
  }
  public void startFunction(View v){
      startService(serviceIn);
  }

  public void stopFunction(View v){
      stopService(serviceIn);
  }
```

```
}
```

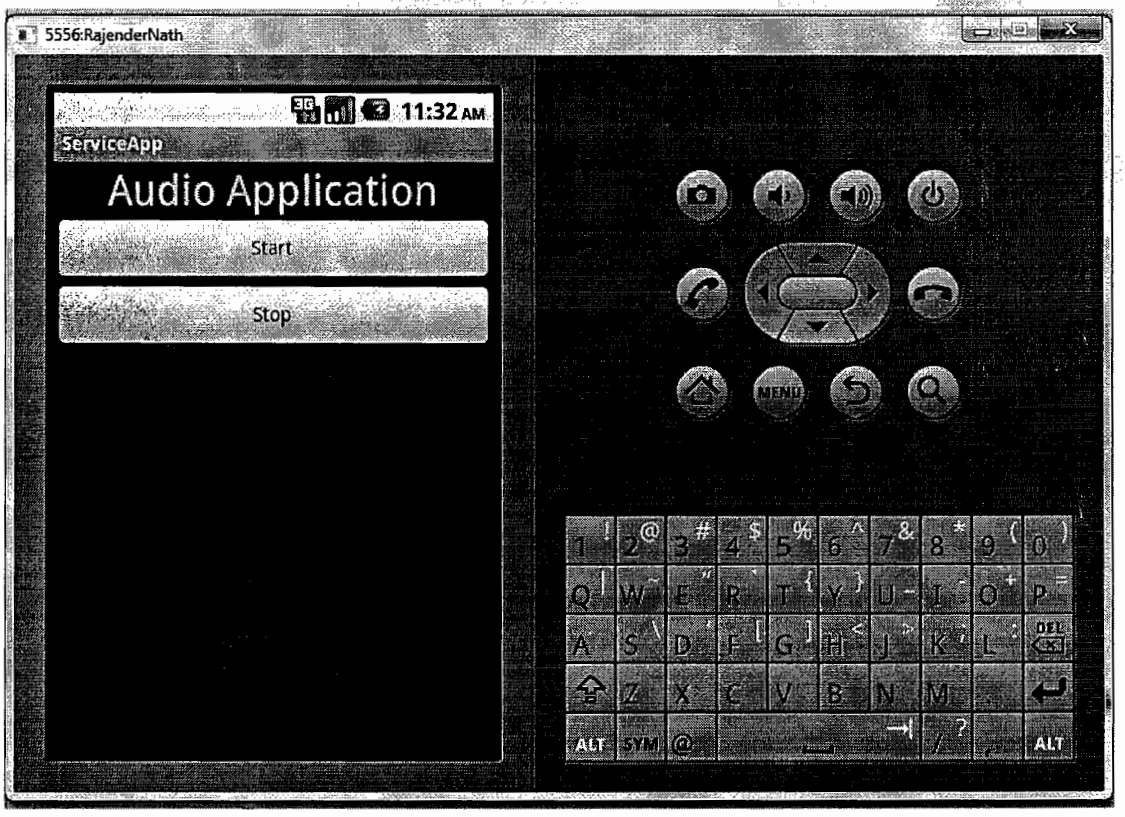
MyService.java

```
package com.rajender.service;

import android.app.Service;
import android.content.Intent;
import android.media.MediaPlayer;
import android.os.IBinder;

public class MyService extends Service {
        MediaPlayer player;
        @Override
        public IBinder onBind(Intent intent){
                return null;
        }
        @Override
        public void onCreate() {
                super.onCreate();
                player=MediaPlayer.create(this, R.raw.vandemataram);
        }
        @Override
        public void onStart(Intent intent, int startId){
                super.onStart(intent, startId);
                player.start();
        }
        @Override
        public void onDestroy() {
                super.onDestroy();
                player.stop();
        }
 }
```

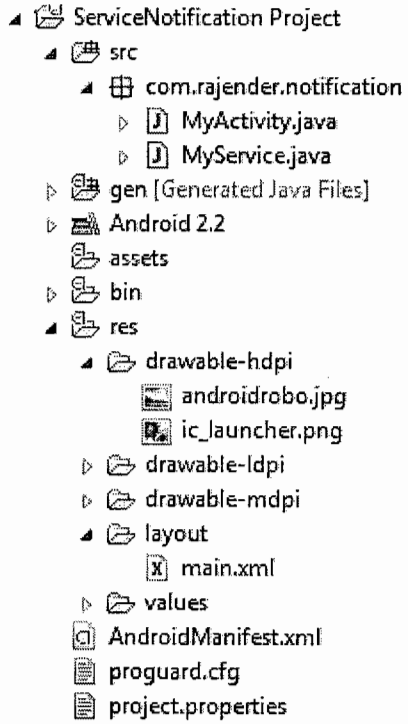
AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
```

```
    package="com.rajender.service"   android:versionCode="1"
    android:versionName="1.0">
   <application
        android:icon="@drawable/icon"  android:label="@string/app_name">
     <activity android:name=".MyActivity"  android:label="@string/app_name">
        <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category ndroid:name="android.intent.category.LAUNCHER" />
        </intent-filter>
     </activity>
     <service android:enabled="true" android:name=".MyService" />
   </application>
  <uses-sdk android:minSdkVersion="8" />
</manifest>
```



Service Notification



main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
   android:layout_width="fill_parent"
   android:layout_height="fill_parent"
   android:orientation="vertical" >

   <TextView
      android:layout_width="fill_parent"
      android:layout_height="wrap_content"
      android:text="Service Notification"
      android:textSize="25px" />

   <Button
      android:onClick="startFunction"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="Start Notification" />

   <Button
            android:onClick="stopFunction"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="Stop Notification" />
```

```
</LinearLayout>
```

MyActivity.java

```
package com.rajender.notification;

import android.app.Activity;
import android.content.Intent;
import android.os.Bundle;
import android.view.View;

public class MyActivity extends Activity {
        Intent serviceIn;
        @Override
   public void onCreate(Bundle b) {
      super.onCreate(b);
      setContentView(R.layout.main);

     serviceIn=new Intent(this,MyService.class);
   }
   public void startFunction(View v){
        startService(serviceIn);
   }

   public void stopFunction(View v){
        stopService(serviceIn);
   }
}
```

MyService.java

```
package com.rajender.notification;

import android.app.Notification;
import android.app.NotificationManager;
import android.app.PendingIntent;
import android.app.Service;
import android.content.Context;
import android.content.Intent;
import android.os.IBinder;

public class MyService  extends Service {
        NotificationManager nm;
        Notification notif;
        PendingIntent pi;
        public static final int ID=123;

        @Override
```

```
public IBinder onBind(Intent intent){
        return null;
}

@Override
public void onCreate() {
        super.onCreate();

        nm=(NotificationManager)getSystemService
                        (Context.NOTIFICATION_SERVICE);

 notif=new Notification(R.drawable.androidrobo,
         "New Message",System.currentTimeMillis());

 Intent intent=new Intent(this,MyService.class);

 pi=PendingIntent.getService(this,0, intent,0);
 notif.setLatestEventInfo(this,"","", pi);
}
        @Override
public void onStart(Intent intent, int startId){
        super.onStart(intent, startId);
        nm.notify(ID,notif);
}
        @Override
public void onDestroy() {
        super.onDestroy();
        nm.cancel(ID);
}
}
```

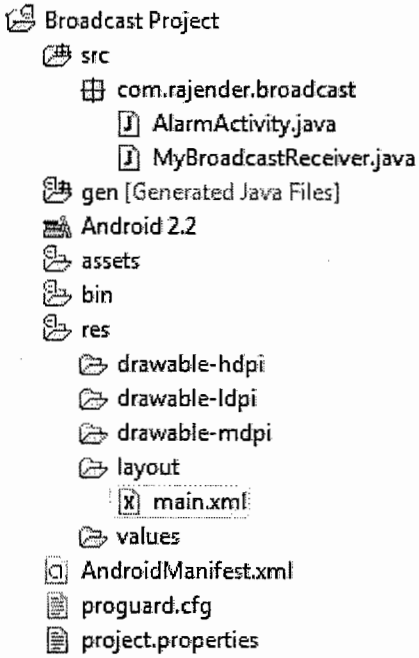
Broadcast receivers

A *broadcast receiver* is a component that does nothing but receives and reacts to broadcast announcements. Many broadcasts originate in system code — for example, announcements that the timezone has changed, that the battery is low, that a picture has been taken, or that the user changed a language preference. Applications can also initiate broadcasts — for example, to let other applications know that some data has been downloaded to the device and is available for them to use.

An application can have any number of broadcast receivers to respond to any announcements it considers important. All receivers extend the BroadcastReceiver base class.

Broadcast receivers do not display a user interface. However, they may start an activity in response to the information they receive, or they may use the NotificationManager to alert the user. Notifications can get the user's attention in various ways — flashing the backlight, vibrating the device, playing a sound, and so on. They typically place a persistent icon in the status bar, which users can open to get the message.

BroadcastExample:-



AlarmActivity.java

```
package com.rajender.broadcast;

import android.app.Activity;
import android.app.AlarmManager;
import android.app.PendingIntent;
import android.content.Intent;
import android.os.Bundle;
import android.view.View;
import android.widget.EditText;
import android.widget.Toast;

public class AlarmActivity extends Activity {
        @Override
        public void onCreate(Bundle savedInstanceState) {
                super.onCreate(savedInstanceState);
                setContentView(R.layout.main);
        }

        public void startAlert(View view) {
                EditText text = (EditText) findViewById(R.id.time);
```

```
                int i = Integer.parseInt(text.getText().toString());
                Intent intent = new Intent(this, MyBroadcastReceiver.class);
                PendingIntent pendingIntent = PendingIntent.getBroadcast(
                                this.getApplicationContext(), 0, intent, 0);
                AlarmManager alarmManager = (AlarmManager)
                                                getSystemService(ALARM_SERVICE);
                alarmManager.set(AlarmManager.RTC_WAKEUP,
                        System.currentTimeMillis()+ (i * 1000), pendingIntent);
                Toast.makeText(this, "Alarm set in " + i + " seconds",
                                Toast.LENGTH_LONG).show();
        }

}
```

MyBroadcastReceiver.java

```
package com.rajender.broadcast;
import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;
import android.os.Vibrator;
import android.widget.Toast;

public class MyBroadcastReceiver extends BroadcastReceiver {
        @Override
        public void onReceive(Context context, Intent intent) {
                Toast.makeText(context,
                                "Please wake up your time is Over!!!!.",
                                Toast.LENGTH_LONG).show();
                // Vibrate the mobile phone
                Vibrator vibrator = (Vibrator)
                        context.getSystemService(Context.VIBRATOR_SERVICE);
                vibrator.vibrate(2000);
        }
}
```

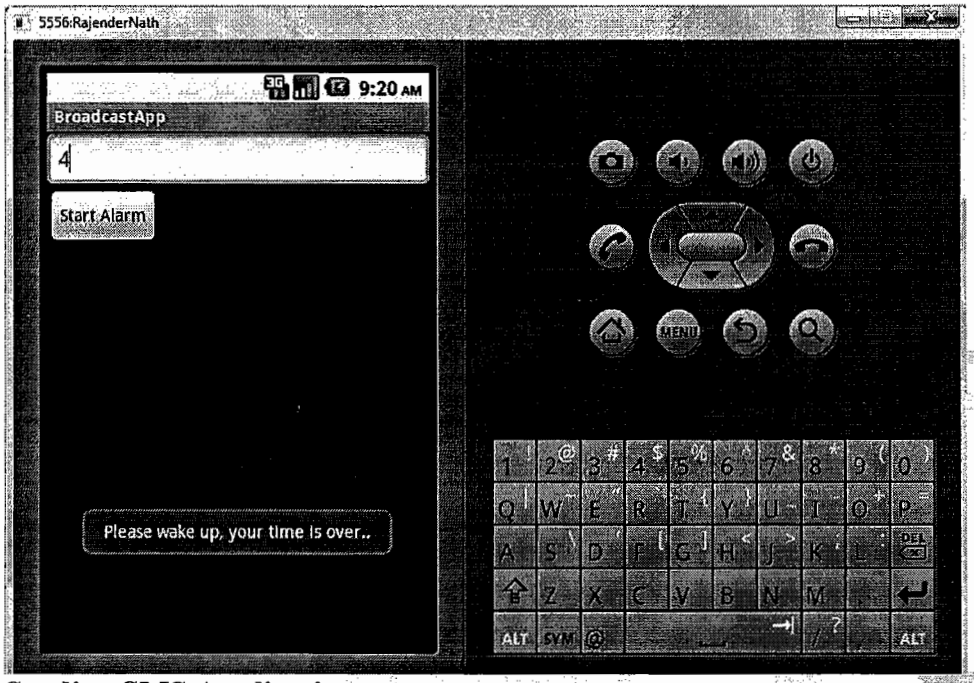
main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
```

```
        android:orientation="vertical" android:layout_width="fill_parent"
        android:layout_height="fill_parent">
        <EditText
                android:layout_height="wrap_content"
                android:id="@+id/time"
                android:layout_width="wrap_content"
                android:hint="Number of seconds"
                android:inputType="numberDecimal"/>
        <Button
                android:text="Start Counter"
                android:onClick="startAlert"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"/>
</LinearLayout>
```

AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.rajender.broadcast"
    android:versionCode="1"
    android:versionName="1.0">
   <uses-sdk android:minSdkVersion="8" />
        <application android:icon="@drawable/icon" android:label="@string/app_name">
                <activity android:name=".AlarmActivity" android:label="@string/app_name">
                        <intent-filter>
                                <action android:name="android.intent.action.MAIN" />
                                <category android:name="android.intent.category.LAUNCHER"
/>
                        </intent-filter>
                </activity>
                <receiver android:name="MyBroadcastReceiver"/>
        </application>
        <uses-permission android:name="android.permission.VIBRATE"/>
</manifest>
```



Sending SMS Application

- SmsSend Project
  - src
    - com.rajender.sms.send
      - SmsSendActivity.java
  - gen [Generated Java Files]
    - com.rajender.sms.send
      - R.java
  - Android 2.2
  - assets
  - bin
  - res
    - drawable-hdpi
    - drawable-ldpi
    - drawable-mdpi
    - layout
      - main.xml
    - values
  - AndroidManifest.xml
  - proguard.cfg
  - project.properties

main.xml

```
<?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"
   android:background="#FFEBCD" >
```

```
<TextView
   android:layout_width="wrap_content"
   android:layout_height="wrap_content"
   android:layout_gravity="center_horizontal"
   android:text="SMS Sending Project"
   android:textSize="30px"
   android:textStyle="bold"
   android:textColor="#f00" />

<TableRow
   android:layout_width="match_parent"
   android:layout_height="wrap_content" >

   <TextView
      android:layout_width="120px"
      android:layout_height="wrap_content"
      android:text="  Phone"
      android:textColor="#00f"
      android:textSize="25px"/>

   <EditText
      android:id="@+id/phoneText"
      android:layout_width="fill_parent"
      android:layout_height="wrap_content" />
</TableRow>

 <TableRow
   android:layout_width="match_parent"
   android:layout_height="wrap_content" >

   <TextView
      android:layout_width="120px"
      android:layout_height="wrap_content"
      android:text="  Message"
      android:textColor="#00f"
      android:textSize="25px"/>

   <EditText
      android:id="@+id/msgText"
      android:layout_width="fill_parent"
      android:layout_height="wrap_content" />
</TableRow>
<Button
   android:onClick="sendSmsFunction"
   android:layout_width="wrap_content"
   android:layout_height="wrap_content"
```

```
    android:text="Send Sms" />

</LinearLayout>
```

SmsSendActivity.java

```
package com.rajender.sms.send;

import android.app.Activity;
import android.app.AlertDialog;
import android.os.Bundle;
import android.telephony.SmsManager;
import android.view.View;
import android.widget.EditText;
import android.widget.Toast;

public class SmsSendActivity extends Activity {
   EditText phone,message;
   String ph,msg;
   SmsManager smsManager;
   AlertDialog.Builder dialog;

        @Override
   public void onCreate(Bundle b) {
      super.onCreate(b);
      setContentView(R.layout.main);
      dialog=new AlertDialog.Builder(this);
      dialog.setNeutralButton("OK",null);

      phone=(EditText)findViewById(R.id.phoneText);
      message=(EditText)findViewById(R.id.msgText);
    }

        public void sendSmsFunction(View v){
                ph = phone.getText().toString().trim();
                msg = message.getText().toString().trim();

                if(ph.equals("")&& msg.equals("")){
                        dialog.setMessage(
                                "Enter Phone number and Message");

                        dialog.show();
                }
```

```
            else if(ph.equals("")){
                    dialog.setMessage("Enter Phone number");
                    dialog.show();
            }
            else if(msg.equals("")){
                    dialog.setMessage("Enter Message");
                    dialog.show();
            }
            else{
            smsManager=SmsManager.getDefault();

            smsManager.sendTextMessage(ph,null,msg,null, null);
            Toast.makeText(getApplicationContext(),
                            "Your message has been sent",
                            Toast.LENGTH_LONG).show();
            }
        }
}
```

AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
   package="com.rajender.sms.send"   android:versionCode="1"
   android:versionName="1.0" >

   <uses-sdk android:minSdkVersion="8" />
   <uses-permission android:name="android.permission.SEND_SMS"/>
   <application
      android:icon="@drawable/ic_launcher"
      android:label="@string/app_name" >
      <activity
         android:name="com.rajender.sms.send.SmsSendActivity"
         android:label="@string/app_name" >
         <intent-filter>
            <action android:name="android.intent.action.MAIN" />

            <category android:name="android.intent.category.LAUNCHER" />
         </intent-filter>
      </activity>
   </application>

</manifest>
```

Sms Receive Project

- SmsReceive Project
  - src
    - com.rajender.sms.receive
      - MySmsReceiver.java
      - SmsReceiveActivity.java
  - gen [Generated Java Files]
  - Android 2.2
  - assets
  - bin
  - res
    - drawable-hdpi
    - drawable-ldpi
    - drawable-mdpi
    - layout
      - main.xml
    - values
  - AndroidManifest.xml
  - proguard.cfg
  - project.properties

main.xml

```
<?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"
    android:background="#FAFAD2">

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="center_horizontal"
        android:text="SMS Receive Project"
        android:textSize="30px"
        android:textColor="#f00"/>

    <TextView
        android:id="@+id/msgTextView"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textColor="#00f"
        android:textSize="20px" />
</LinearLayout>
```

SmsReceiveActivity.java

```
package com.rajender.sms.receive;

import android.app.Activity;
import android.os.Bundle;
import android.widget.TextView;

public class SmsReceiveActivity extends Activity{
    public static TextView msgText;
        @Override
    public void onCreate(Bundle b) {
        super.onCreate(b);
        setContentView(R.layout.main);
        msgText=(TextView)findViewById(R.id.msgTextView);

    }
        public static void displayMessage(String str){
                msgText.append(str);
        }
}
```

MySmsReceiver.java

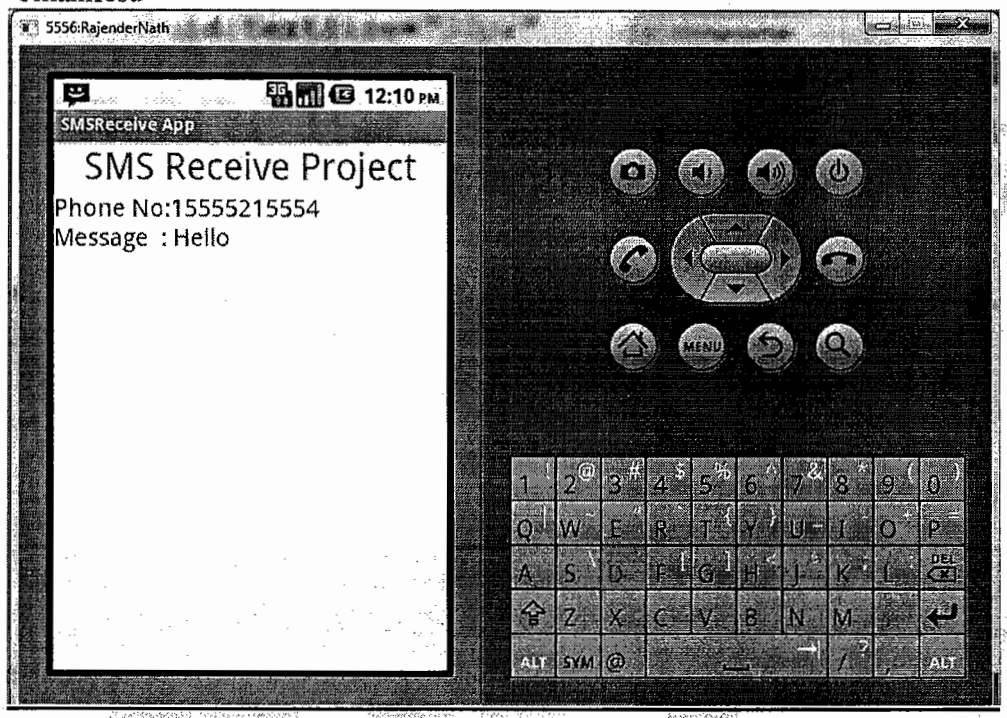
```
package com.rajender.sms.receive;
```

```
import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;
import android.os.Bundle;
import android.telephony.SmsMessage;

public class MySmsReceiver extends BroadcastReceiver {
        @Override
        public void onReceive(Context context,Intent intent) {
                Bundle bundle= intent.getExtras();
                Object[] messages=(Object[])bundle.get("pdus");
                SmsMessage[] saveMsg=new SmsMessage[messages.length];

                for(int i=0;i<messages.length;i++){
                saveMsg[i] =SmsMessage.createFromPdu((byte[])messages[i]);
                }

                for(SmsMessage msg:saveMsg){
                        String phone=msg.getOriginatingAddress();
                        String message=msg.getMessageBody();
                        SmsReceiveActivity.displayMessage(
                        "Phone No:"+phone+"\nMessage  : "+message+"\n\n");
                }
        }
}
```

AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.rajender.sms.receive"   android:versionCode="1"
    android:versionName="1.0" >
  <uses-sdk android:minSdkVersion="8" />
 <uses-permission android:name="android.permission.RECEIVE_SMS"/>
<application
   android:icon="@drawable/ic_launcher"  android:label="@string/app_name" >

      <activity
        android:name="SmsReceiveActivity"
        android:label="@string/app_name" >
        <intent-filter>
          <action android:name="android.intent.action.MAIN" />
          <category android:name="android.intent.category.LAUNCHER" />
        </intent-filter>
      </activity>
       <receiver android:name="MySmsReceiver">
       <intent-filter>
```

```
    <action android:name="android.provider.Telephony.SMS_RECEIVED"/>
  </intent-filter>
 </receiver>
 </application>

</manifest>
```



Content providers

A *content provider* makes a specific set of the application's data available to other applications. The data can be stored in the file system, in an SQLite database, or in any other manner that makes sense. The content provider extends the ContentProvider base class to implement a standard set of methods that enable other applications to retrieve and store data of the type it controls. However, applications do not call these methods directly. Rather they use a ContentResolver object and call its methods instead. A ContentResolver can talk to any content provider; it cooperates with the provider to manage any interprocess communication that's involved.

Whenever there's a request that should be handled by a particular component, Android makes sure that the application process of the component is running, starting it if necessary, and that an appropriate instance of the component is available, creating the instance if necessary.

Activating components: intents

Content providers are activated when they're targeted by a request from a ContentResolver. The other three components — activities, services, and broadcast receivers — are activated by asynchronous messages called *intents*. An intent is an Intent object that holds the content of the message. For activities and services, it names the action being requested and specifies the URI of the data to act on, among other things. For example, it might convey a request for an activity to present an image to the user or let the user edit some text. For broadcast receivers, the Intent object names the action being announced. For example, it might announce to interested parties that the camera button has been pressed.

There are separate methods for activating each type of component:

- An activity is launched (or given something new to do) by passing an Intent object to Context.startActivity() or Activity.startActivityForResult(). The responding activity can look at the initial intent that caused it to be launched by calling its getIntent() method. Android calls the activity's onNewIntent() method to pass it any subsequent intents.

  One activity often starts the next one. If it expects a result back from the activity it's starting, it calls startActivityForResult() instead of startActivity(). For example, if it starts an activity that lets the user pick a photo, it might expect to be returned the chosen photo. The result is returned in an Intent object that's passed to the calling activity's onActivityResult() method.

- A service is started (or new instructions are given to an ongoing service) by passing an Intent object to Context.startService(). Android calls the service's onStart() method and passes it the Intent object.

  Similarly, an intent can be passed to Context.bindService() to establish an ongoing connection between the calling component and a target service. The service receives the Intent object in an onBind() call. (If the service is not already running, bindService() can optionally start it.) For example, an activity might establish a connection with the music playback service mentioned earlier so that it can provide the user with the means (a user interface) for controlling the playback. The activity would call bindService() to set up that connection, and then call methods defined by the service to affect the playback.

- An application can initiate a broadcast by passing an Intent object to methods like Context.sendBroadcast(), Context.sendOrderedBroadcast(), and Context.sendStickyBroadcast() in any of their variations. Android delivers the intent to all interested broadcast receivers by calling their onReceive() methods.

Shutting down components

A content provider is active only while it's responding to a request from a ContentResolver. And a broadcast receiver is active only while it's responding to a broadcast message. So there's no need to explicitly shut down these components.

Activities, on the other hand, provide the user interface. They're in a long-running conversation with the user and may remain active, even when idle, as long as the conversation continues. Similarly, services may also remain running for a long time. So Android has methods to shut down activities and services in an orderly way:

- An activity can be shut down by calling its finish() method. One activity can shut down another activity (one it started with startActivityForResult()) by calling finishActivity().
- A service can be stopped by calling its stopSelf() method, or by calling Context.stopService().

The manifest file

Before Android can start an application component, it must learn that the component exists. Therefore, applications declare their components in a manifest file that's bundled into the Android package, the .apk file that also holds the application's code, files, and resources.

The manifest is a structured XML file and is always named AndroidManifest.xml for all applications. It does a number of things in addition to declaring the application's components, such as naming any libraries the application needs to be linked against (besides the default Android library) and identifying any permissions the application expects to be granted.

But the principal task of the manifest is to inform Android about the application's components. For example, an activity might be declared as follows:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest . . . >
  <application . . . >
    <activity android:name="com.example.project.FreneticActivity"
              android:icon="@drawable/small_pic.png"
              android:label="@string/freneticLabel"
              . . . >
    </activity>
    . . .
  </application>
</manifest>
```

The name attribute of the <activity> element names the Activity subclass that implements the activity. The icon and label attributes point to resource files containing an icon and label that can be displayed to users to represent the activity.

The other components are declared in a similar way — <service> elements for services, <receiver> elements for broadcast receivers, and <provider> elements for content providers. Activities, services, and content providers that are not declared in the manifest are not visible to the system and are consequently never run. However, broadcast receivers can either be declared in the manifest, or they can be created dynamically in code (as BroadcastReceiver objects) and registered with the system by calling Context.registerReceiver().

Intents

To establish communication between two applications then we can use Intents.
We can use intents

1. Starting an activity
2. Broadcast intents
3. Services

There are two types of intents.

1. Explicit Intents
2. Implicit Intents

Explicit Intents:-

Intent myIntent=new Intent(Context ,Destination class);
startActivity(myIntent);
Ex:-
onClick(View v){
Intent myIntent=new Intent(getApplicationContext()/FirstActivity.this, SecondActivity.class);
}

Implicit Intents:-

Intent myIntent=new Intent(action, URI, "Data");
Ex:-
Intent myIntent=new Intent(Intent.ACTION_DIAL);
startActivity(myIntent);

Transferring data by using **ExtraBundle** class

If you want to transfer the data one activity to another activity then we use putExtra(key,value); this method has given in ExtraBundle classs.

```
public class NextActivity extends Activity {

//Your member variable declaration here

// Called when the activity is first created.

@Override

public void onCreate(Bundle savedInstanceState) {

//code here

}

}
```

After we have created the new Activity, we have to register it in file 'AndroidManifest.xml'. For registering we have to create an entry in 'AndroidManifest.xml' as

<activity android:name=".NextActivity"
android:label="@string/app_name">

</activity>

Note that here we have not used intent filter, since we are going to use an explicit intent, the syntax of intent filter is

<intent-filter>

<action android:name="<action here>"/>

<category android:name="<category here>"/>

</intent-filter>

Here,
action — The general action to be performed

category — Gives additional information about the action to execute. For example, CATEGORY_LAUNCHER means it should appear in the Launcher as a top-level application, while CATEGORY_ALTERNATIVE means it should be included in a list of alternative actions the user can perform on a piece of data.

Note: you can also use application tab below the **'AndroidManifest.xml'** file, and in **'Application Nodes'** section click
'Add' button as shown in figure below and select the activity .

Next you can start this activity on any event as follows

Intent myIntent = new Intent(CurrentActivity.this, NextActivity.class);
CurrentActivity.this.startActivity(myIntent);

Here, you have to create and intent with **CurrentActivity.this** as first parameter and the the **next activity** as second parameter.

After you have created the intent, you can start the new activity by calling **startActivity**, on current Activity, with the created intent as parameter

Different types of Actions/Activity actions

1. ACTION_MAIN :- It defines the main entry point of the activity.(Just like a main() method).

   Ex:-android.intent.action.MAIN (the value of the attribute).
2. ACTION_VIEW :- It displays the data to the user. android.intent.action.VIEW
3. ACTION_EDIT :- It provides explicit editable access to the given data.
4. ACTION_PICK :- It picks an item from data. It will return the selected item. Ex: Contact Book--->pick--->CALL/DIAL
5. ACTION_DIAL :- It defined a number which is specified by the data. It displays a user interface with the number to be dialed allowing the user to explicitly start the call.
6. ACTION_CALL :- It initiate a direct call to the number specified.
7. ACTION_ANSWER :-it handles an incoming phone call.
8. ACTION_SEARCH :- It performs a search operation for the defined value. A search manager is used to search the data.
9. ACTION_DATA :-It insert an empty item into the container.
10. ACTION_SYNC :-it performs data synchronization with the other activity.

Standard Categories

1. CATEGORY_LAUNCHER:-The Activity defines with this category should be displayed in the top level launcher.
2. CATEGORY_DEFAULT:-This category defines if the activity should be an option for the default action.
3. CATEGORY_CAR_DOCK:-It defines an activity to run when a device is inserted in a car dock. For cars.
4. CATEGORY_CAR_MODE:-It is used to indicates that the action can be used in a car environment.
5. CATEGORY_TAB:-It is supposed to be used as a tab inside a TabActivity(predefined class).
6. CATEGORY_INFO:- It provides the Information about the package it is placed into it.

Standard Broadcast Actions

1. ACTION_BATERY_CHANGED :-This defines a Bradcast intent containing the charging state changing level about the battery. This is maintained by the Battery Manager.
2. ACTION_BATERY_LOW :- It indicates low battery conditions on the device. Ex:- It is a Dialog box.
3. ACTION_BATERY_OK:- It indicates the batteries now ok with the charging level.
4. ACTION_TIME_CHANGED:- It indicates the time is set and it pose in a Toast message.

How to create a new Android activity

The Android development paradigm introduces for the developer the "Activity": <> During development you will find it easy to wrap-up parts of the functionality of your code into independent activities, with their own life cycle. More about activity life cycle on the Android developers website.

Easy (just start activity)
Assume you have the MainActivity, and a SecondaryActivity you want to start, this may be done in the easiest way:

```
Intent intent = new Intent(Intent.ACTION_VIEW);
intent.setClassName(this, SecondaryActivity.class.getName());
startActivity(intent);
```

Medium (pass some params as well)
Now imagine you want to pass some data to the new intent. You will achieve this using the Bundle class, to encapsulate your data.

Example:

```
Intent intent = new Intent();
Bundle bun = new Bundle();

bun.putString("param_string", "the actual string"); // add two parameters: a string and a boolean
bun.putBoolean("param_bool", true);

intent.setClass(this, SecondaryActivity.class);
intent.putExtras(bun);
startActivity(intent);
```

In the SecondaryActivity, you will need to access these params. This is how:

```
Bundle bun = getIntent().getExtras();
String param1 = bun.getString("param_string");
boolean param2 = bun.getBoolean("param_bool");
```

Some useful default activities:
Part of the beauty of Android is that you can encapsulate other functionality/activities in your own code, just by writing a few lines of code. Below is a list of useful activities one may consider to use:

1. Writing an email

```
Intent intent = new Intent(Intent.ACTION_SENDTO,
Uri.parse("mailto:office@example.com"));
intent.putExtra("subject", "my subject");
intent.putExtra("body", "my message");
startActivity(intent);
```

2. Browse to a web-page

```
Intent intent = new Intent(Intent.ACTION_VIEW, Uri.parse("http://www.google.com"));
startActivity(intent);
```

3. Write a SMS

```
Intent intent = new Intent(Intent.ACTION_SENDTO, Uri.parse("sms://"));
intent.putExtra("address", "");
intent.putExtra("sms_body", "my message");
startActivity(intent);
```

4. Search something on Google

```
Intent intent = new Intent(Intent.ACTION_WEB_SEARCH );
intent.putExtra(SearchManager.QUERY, "search this text");
startActivity(intent);
```

5. **Get the Wiktionary of some word**

```
Intent intent = new Intent(Intent.ACTION_VIEW, Uri.parse("http://en.wiktionary.org/wiki/" +
"word"));
startActivity(intent);
```

6. Get the Wikipedia page of some words

```
String uri = "http://en.wikipedia.org/wiki/" + "my text";
Intent intent = new Intent(Intent.ACTION_VIEW, Uri.parse(uri));
startActivity(intent);
```

Intent filters

An Intent object can explicitly name a target component. If it does, Android finds that component (based on the declarations in the manifest file) and activates it. But if a target is not explicitly named, Android must locate the best component to respond to the intent. It does so by comparing the Intent object to the *intent filters* of potential targets. A component's intent filters inform Android of the kinds of intents the component is able to handle. Like other essential information about the component, they're declared in the manifest file. Here's an extension of the previous example that adds two intent filters to the activity:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest . . . >
  <application . . . >
    <activity android:name="com.example.project.FreneticActivity"
          android:icon="@drawable/small_pic.png"
          android:label="@string/freneticLabel"
          . . . >
      <intent-filter . . . >
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
      </intent-filter>
      <intent-filter . . . >
        <action android:name="com.example.project.BOUNCE" />
        <data android:mimeType="image/jpeg" />
        <category android:name="android.intent.category.DEFAULT" />
      </intent-filter>
    </activity>
    . . .
  </application>
</manifest>
```

The first filter in the example — the combination of the action "android.intent.action.MAIN" and the category "android.intent.category.LAUNCHER" — is a common one. It marks the activity as one that should be represented in the application launcher, the screen listing applications users can launch on the device. In other words, the activity is the entry point for the application, the initial one users would see when they choose the application in the launcher.

The second filter declares an action that the activity can perform on a particular type of data.

A component can have any number of intent filters, each one declaring a different set of capabilities. If it doesn't have any filters, it can be activated only by intents that explicitly name the component as the target.

For a broadcast receiver that's created and registered in code, the intent filter is instantiated directly as an IntentFilter object. All other filters are set up in the manifest.

Creating Dialogs

A dialog is a small window that appears in front of the current Activity. The underlying Activity loses focus and the dialog accepts all user interaction. Dialogs are normally used for notifications and short activities that directly relate to the application in progress.

The Android API supports the following types of Dialog objects:

AlertDialog
: A dialog that can manage zero, one, two, or three buttons, and/or a list of selectable items that can include checkboxes or radio buttons. The AlertDialog is capable of constructing most dialog user interfaces and is the suggested dialog type.

ProgressDialog
: A dialog that displays a progress wheel or progress bar. Because it's an extension of the AlertDialog, it also supports buttons.

DatePickerDialog
: A dialog that allows the user to select a date.

TimePickerDialog
: A dialog that allows the user to select a time.

If you would like to customize your own dialog, you can extend the base Dialog object or any of the subclasses listed above and define a new layout.

Showing a Dialog

A dialog is always created and displayed as a part of an Activity. You should normally create dialogs from within your Activity's onCreateDialog(int) callback method. When you use this callback, the Android system automatically manages the state of each dialog and hooks them to the Activity, effectively making it the "owner" of each dialog. As such, each dialog inherits certain properties from the Activity. For example, when a dialog is open, the Menu key reveals the options menu defined for the Activity and the volume keys modify the audio stream used by the Activity.

Note: If you decide to create a dialog outside of the onCreateDialog() method, it will not be attached to an Activity. You can, however, attach it to an Activity with setOwnerActivity(Activity).

When you want to show a dialog, call showDialog(int) and pass it an integer that uniquely identifies the dialog that you want to display.

When a dialog is requested for the first time, Android calls onCreateDialog(int) from your Activity, which is where you should instantiate the Dialog. This callback method is passed the same ID that you passed to showDialog(int). After you create the Dialog, return the object at the end of the method.

Before the dialog is displayed, Android also calls the optional callback method onPrepareDialog(int, Dialog). Define this method if you want to change any properties of the dialog each time it is opened. This method is called every time a dialog is opened, whereas onCreateDialog(int) is only called the very first time a dialog is opened. If you don't define onPrepareDialog(), then the dialog will remain the same as it was the previous time it was opened. This method is also passed the dialog's ID, along with the Dialog object you created in onCreateDialog().

The best way to define the onCreateDialog(int) and onPrepareDialog(int, Dialog) callback methods is with a *switch* statement that checks the *id* parameter that's passed into the method. Each *case* should check for a unique dialog ID and then create and define the respective Dialog. For example, imagine a game that uses two different dialogs: one to indicate that the game has paused and another to indicate that the game is over. First, define an integer ID for each dialog:

```
static final int DIALOG_PAUSED_ID = 0;
static final int DIALOG_GAMEOVER_ID = 1;
```

Then, define the onCreateDialog(int) callback with a switch case for each ID:

```
protected Dialog onCreateDialog(int id) {
   Dialog dialog;
   switch(id) {
   case DIALOG_PAUSED_ID:
      // do the work to define the pause Dialog
      break;
   case DIALOG_GAMEOVER_ID:
      // do the work to define the game over Dialog
      break;
   default:
      dialog = null;
   }
   return dialog;
}
```

When it's time to show one of the dialogs, call showDialog(int) with the ID of a dialog:

```
showDialog(DIALOG_PAUSED_ID);
```

Dismissing a Dialog

When you're ready to close your dialog, you can dismiss it by calling dismiss() on the Dialog object. If necessary, you can also call dismissDialog(int) from the Activity, which effectively calls dismiss() on the Dialog for you.

If you are using onCreateDialog(int) to manage the state of your dialogs (as discussed in the previous section), then every time your dialog is dismissed, the state of the Dialog object is retained by the Activity. If you decide that you will no longer need this object or it's important that the state is cleared, then you should call removeDialog(int). This will remove any internal references to the object and if the dialog is showing, it will dismiss it.

Using dismiss listeners

If you'd like your applcation to perform some procedures the moment that a dialog is dismissed, then you should attach an on-dismiss listener to your Dialog.

First define the DialogInterface.OnDismissListener interface. This interface has just one method, onDismiss(DialogInterface), which will be called when the dialog is dismissed. Then simply pass your OnDismissListener implementation to setOnDismissListener().

However, note that dialogs can also be "cancelled." This is a special case that indicates the dialog was explicitly cancelled by the user. This will occur if the user presses the "back" button to close the dialog, or if the dialog explicitly calls cancel() (perhaps from a "Cancel" button in the dialog). When a dialog is cancelled, the OnDismissListener will still be notified, but if you'd like to be informed that the dialog was explicitly cancelled (and not dismissed normally), then you should register an DialogInterface.OnCancelListener with setOnCancelListener().

Creating an AlertDialog

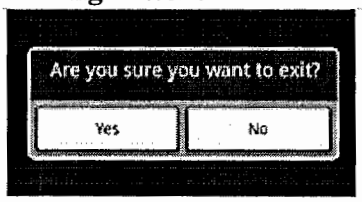
An AlertDialog is an extension of the Dialog class. It is capable of constructing most dialog user interfaces and is the suggested dialog type. You should use it for dialogs that use any of the following features:

- A title
- A text message
- One, two, or three buttons
- A list of selectable items (with optional checkboxes or radio buttons)

To create an AlertDialog, use the AlertDialog.Builder subclass. Get a Builder with AlertDialog.Builder(Context) and then use the class's public methods to define all of the AlertDialog properties. After you're done with the Builder, retrieve the AlertDialog object with create().

The following topics show how to define various properties of the AlertDialog using the AlertDialog.Builder class. If you use any of the following sample code inside your onCreateDialog() callback method, you can return the resulting Dialog object to display the dialog.

Adding buttons



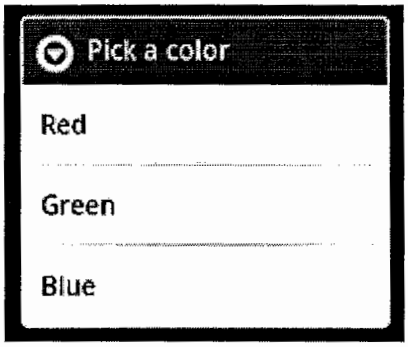
To create an AlertDialog with side-by-side buttons like the one shown in the screenshot to the right, use the set...Button() methods:

```
AlertDialog.Builder builder = new AlertDialog.Builder(this);
builder.setMessage("Are you sure you want to exit?")
    .setCancelable(false)
    .setPositiveButton("Yes", new DialogInterface.OnClickListener() {
       public void onClick(DialogInterface dialog, int id) {
          MyActivity.this.finish();
       }
    })
    .setNegativeButton("No", new DialogInterface.OnClickListener() {
       public void onClick(DialogInterface dialog, int id) {
          dialog.cancel();
       }
    });
AlertDialog alert = builder.create();
```

First, add a message for the dialog with setMessage(CharSequence). Then, begin method-chaining and set the dialog to be *not cancelable* (so the user cannot close the dialog with the back button) with setCancelable(boolean). For each button, use one of the set...Button() methods, such as setPositiveButton(), that accepts the name for the button and a DialogInterface.OnClickListener that defines the action to take when the user selects the button.

Note: You can only add one of each button type to the AlertDialog. That is, you cannot have more than one "positive" button. This limits the number of possible buttons to three: positive, neutral, and negative. These names are technically irrelevant to the actual functionality of your buttons, but should help you keep track of which one does what.

Adding a list

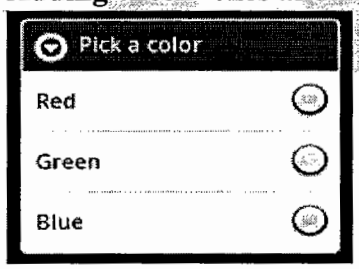


To create an AlertDialog with a list of selectable items like the one shown to the right, use the setItems() method:

```
final CharSequence[] items = {"Red", "Green", "Blue"};

AlertDialog.Builder builder = new AlertDialog.Builder(this);
builder.setTitle("Pick a color");
builder.setItems(items, new DialogInterface.OnClickListener() {
    public void onClick(DialogInterface dialog, int item) {
        Toast.makeText(getApplicationContext(), items[item], Toast.LENGTH_SHORT).show();
    }
});
AlertDialog alert = builder.create();
```

First, add a title to the dialog with setTitle(CharSequence). Then, add a list of selectable items with setItems(), which accepts the array of items to display and a DialogInterface.OnClickListener that defines the action to take when the user selects an item.

Adding checkboxes and radio buttons



To create a list of multiple-choice items (checkboxes) or single-choice items (radio buttons) inside the dialog, use the setMultiChoiceItems() and setSingleChoiceItems() methods, respectively. If you create one of these selectable lists in the onCreateDialog() callback method, Android manages the state of the list for you. As long as the Activity is active, the dialog remembers the items that were previously selected, but when the user exits the Activity, the selection is lost.

Note: To save the selection when the user leaves or pauses the Activity, you must properly save and restore the setting throughout the Activity Lifecycle. To permanently save the selections,

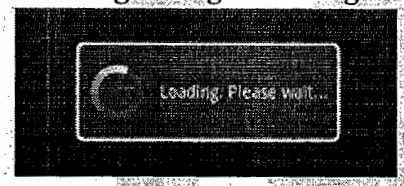
even when the Activity process is completely shutdown, you need to save the settings with one of the Data Storage techniques.

To create an AlertDialog with a list of single-choice items like the one shown to the right, use the same code from the previous example, but replace the setItems() method with setSingleChoiceItems():

```
final CharSequence[] items = {"Red", "Green", "Blue"};

AlertDialog.Builder builder = new AlertDialog.Builder(this);
builder.setTitle("Pick a color");
builder.setSingleChoiceItems(items, -1, new DialogInterface.OnClickListener() {
   public void onClick(DialogInterface dialog, int item) {
      Toast.makeText(getApplicationContext(), items[item], Toast.LENGTH_SHORT).show();
   }
});
AlertDialog alert = builder.create();
```

The second parameter in the setSingleChoiceItems() method is an integer value for the *checkedItem*, which indicates the zero-based list position of the default selected item. Use "-1" to indicate that no item should be selected by default.

Creating a ProgressDialog



A ProgressDialog is an extension of the AlertDialog class that can display a progress animation in the form of a spinning wheel, for a task with progress that's undefined, or a progress bar, for a task that has a defined progression. The dialog can also provide buttons, such as one to cancel a download.

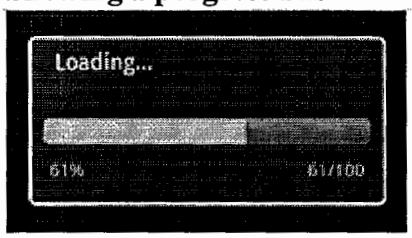
Opening a progress dialog can be as simple as calling ProgressDialog.show(). For example, the progress dialog shown to the right can be easily achieved without managing the dialog through the onCreateDialog(int) callback, as shown here:

```
ProgressDialog dialog = ProgressDialog.show(MyActivity.this, "",
                "Loading. Please wait...", true);
```

The first parameter is the application Context, the second is a title for the dialog (left empty), the third is the message, and the last parameter is whether the progress is indeterminate (this is only relevant when creating a progress bar, which is discussed in the next section).

The default style of a progress dialog is the spinning wheel. If you want to create a progress bar that shows the loading progress with granularity, some more code is required, as discussed in the next section.

Showing a progress bar



To show the progression with an animated progress bar:

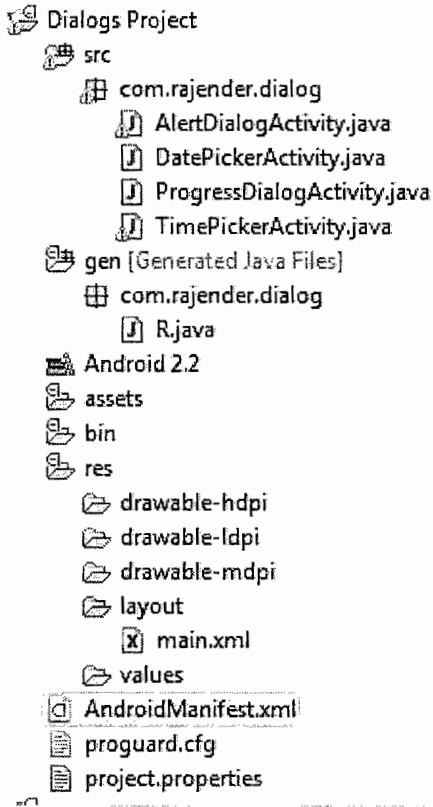
1. Initialize the ProgressDialog with the class constructor, ProgressDialog(Context).
2. Set the progress style to "STYLE_HORIZONTAL" with setProgressStyle(int) and set any other properties, such as the message.
3. When you're ready to show the dialog, call show() or return the ProgressDialog from the onCreateDialog(int) callback.
4. You can increment the amount of progress displayed in the bar by calling either setProgress(int) with a value for the total percentage completed so far or incrementProgressBy(int) with an incremental value to add to the total percentage completed so far.

For example, your setup might look like this:

```
ProgressDialog progressDialog;
progressDialog = new ProgressDialog(mContext);
progressDialog.setProgressStyle(ProgressDialog.STYLE_HORIZONTAL);
progressDialog.setMessage("Loading...");
progressDialog.setCancelable(false);
```

The setup is simple. Most of the code needed to create a progress dialog is actually involved in the process that updates it. You might find that it's necessary to create a second thread in your application for this work and then report the progress back to the Activity's UI thread with a Handler object. If you're not familiar with using additional threads with a Handler, see the example Activity below that uses a second thread to increment a progress dialog managed by the Activity.

Dialogs Project :-



AlertDialog Ex:-

main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
   android:layout_width="fill_parent"
   android:layout_height="fill_parent"
   android:orientation="vertical" >

   <TextView
      android:layout_width="fill_parent"
      android:layout_height="wrap_content"
      android:text="Dialog Application"
      android:textSize="25px"
      android:textColor="#f00" />

   <Button
      android:onClick="dialogFunction"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="Show Dialog"
      android:textSize="20px" />

</LinearLayout>
```

AlertDialogActivity.java

```
package com.rajender.dialog;

import android.app.Activity;
import android.app.AlertDialog;
import android.content.DialogInterface;
import android.os.Bundle;
import android.view.View;
import android.widget.Toast;

 public class AlertDialogActivity extends Activity {
        @Override
   public void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.alertdialog);
        }

public void dialogFunction (View v){
        AlertDialog.Builder alertbox = new AlertDialog.Builder(this);
     // set the message to display
      alertbox.setMessage("This is the alertbox!");
     alertbox.setTitle("Alert Msg");
     alertbox.setNeutralButton("Cancel", new DialogInterface.OnClickListener() {
       // click listener on the alert box
          public void onClick(DialogInterface d, int id) {
             // the button was clicked
             Toast.makeText(getApplicationContext(),
                   "Cancel button clicked", Toast.LENGTH_LONG).show();
          }
       });
         alertbox.setPositiveButton("Yes", new DialogInterface.OnClickListener() {
           // click listener on the alert box
          public void onClick(DialogInterface d, int id) {
             // the button was clicked
             Toast.makeText(getApplicationContext(),
```

```
                "Yes button clicked", Toast.LENGTH_LONG).show();
            }
        });
         alertbox.setNegativeButton("No", new DialogInterface.OnClickListener() {
            // click listener on the alert box
            public void onClick(DialogInterface d, int id) {
               // the button was clicked
               Toast.makeText(getApplicationContext(),
                        "No button clicked", Toast.LENGTH_LONG).show();
            }
        });

       alertbox.setCancelable(true);
        alertbox.show();

        }
}
```

ProgressDialog Ex:-

ProgressDialogActivity.java

```
package com.rajender.dialog;

import android.app.Activity;
import android.app.ProgressDialog;
import android.os.Bundle;
import android.view.View;

public class ProgressDialogActivity extends Activity {

        @Override
    public void onCreate(Bundle savedInstanceState) {
       super.onCreate(savedInstanceState);
       setContentView(R.layout.progressdialog);
        }

public void dialogFunction (View v){
```

```
        ProgressDialog dialog = new ProgressDialog(this);
       // make the progress bar cancel
        dialog.setCancelable(true);
        // set a message text
        dialog.setMessage("Loading...");

        //Setting the styles as horizontal
       // dialog.setProgressStyle(ProgressDialog.STYLE_HORIZONTAL);

        //setting style as spinning
        //dialog.setProgressStyle(ProgressDialog.STYLE_SPINNER);
        // show it
        dialog.show();

                }

        }
```

DatePickerDialog Ex:-

DatePickerActivity.java

```
package com.rajender.dialog;

import java.util.Calendar;
import android.app.Activity;
import android.app.DatePickerDialog;
import android.os.Bundle;
import android.view.View;
import android.widget.DatePicker;
import android.widget.Toast;

public class DatePickerActivity extends Activity
 {
        @Override
        protected void onCreate(Bundle b) {
                super.onCreate(b);
                setContentView(R.layout.main);
```

```
    }

    public void dialogFunction(View v){
        //Get the System date
        Calendar c=Calendar.getInstance();
        int cyear=c.get(Calendar.YEAR);
        int cmonth=c.get(Calendar.MONTH);
        int cday=c.get(Calendar.DAY_OF_MONTH);

        //create DatePickerDialog object
        DatePickerDialog dialog=
                    new DatePickerDialog(this,
                    dateList, cyear,cmonth,cday);
                dialog.show();
    }
    //Create Listener object and handle event
    DatePickerDialog.OnDateSetListener dateList
     =new DatePickerDialog.OnDateSetListener(){

        @Override
        public void onDateSet(DatePicker view,
                              int year, int month,int day) {
            //Provide Business logic
            String date=month+1+"-"+day+"-"+year;

            Toast.makeText(getApplicationContext(),
                        "Selected Date is : "+date,
                        Toast.LENGTH_LONG).show();
        }
    };
}
```

TimePickerDialog Ex:-

TimePickerAcitivity.java

```
package com.rajender.dialog;

import java.util.Calendar;
```

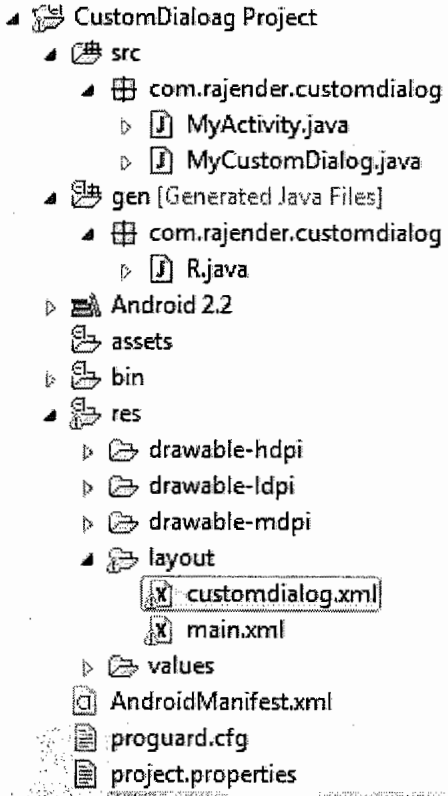
```
import android.app.Activity;
import android.app.TimePickerDialog;
import android.os.Bundle;
import android.view.View;
import android.widget.TimePicker;
import android.widget.Toast;


public class TimePickerActivity extends Activity
 {
        @Override
        protected void onCreate(Bundle b) {
                super.onCreate(b);
                setContentView(R.layout.main);
        }

        public void dialogFunction(View v){
                //Get the System date
                Calendar c=Calendar.getInstance();
                int chour=c.get(Calendar.HOUR_OF_DAY);
                int cminute=c.get(Calendar.MINUTE);

                //create TimePickerDialog object
                TimePickerDialog dialog=
                        new TimePickerDialog(this, timeList, chour,cminute,false);
                        dialog.show();
        }
        //Create Listener object and handle event
        TimePickerDialog.OnTimeSetListener timeList
          =new TimePickerDialog.OnTimeSetListener(){

                @Override
                public void onTimeSet(TimePicker view, int hour, int minute) {
                        //Provide Business logic
                        String time=hour+" : "+minute;

                        Toast.makeText(getApplicationContext( ), "Selected Time is : "+time,
                                        Toast.LENGTH_LONG).show();
                }
        };
 }
```

AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
   package="com.rajender.dialog" android:versionCode="1" android:versionName="1.0" >
   <uses-sdk android:minSdkVersion="8" />
```

```
<application
  android:icon="@drawable/ic_launcher" android:label="@string/app_name" >
   <activity  android:name=".AlertDialogActivity"
        android:label="@string/app_name" >
      <intent-filter>
         <action android:name="android.intent.action.MAIN" />
         <category android:name="android.intent.category.LAUNCHER" />
      </intent-filter>
   </activity>
     <activity  android:name=".DatePickerActivity">
      <intent-filter>
         <action android:name="android.intent.action.EDIT" />
         <category android:name="android.intent.category.DEFAULT" />
      </intent-filter>
   </activity>
   <activity  android:name="TimePickerActivity" >
      <intent-filter>
        <action android:name="android.intent.action.EDIT" />
        <category android:name="android.intent.category.DEFAULT" />
      </intent-filter>
   </activity>
    <activity  android:name="ProgressDialogActivity" >
      <intent-filter>
         <action android:name="android.intent.action.EDIT" />
         <category android:name="android.intent.category.DEFAULT" />
      </intent-filter>
   </activity>
  </application>
</manifest>
```

Custom Dialog Example :-



main.xml

```
<?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"
   android:background="#00ffff" >

   <Button
      android:onClick="showCustomDialog"
      android:layout_width="fill_parent"
      android:layout_height="wrap_content"
      android:text="Show Custom Dialog" />

</LinearLayout>
```

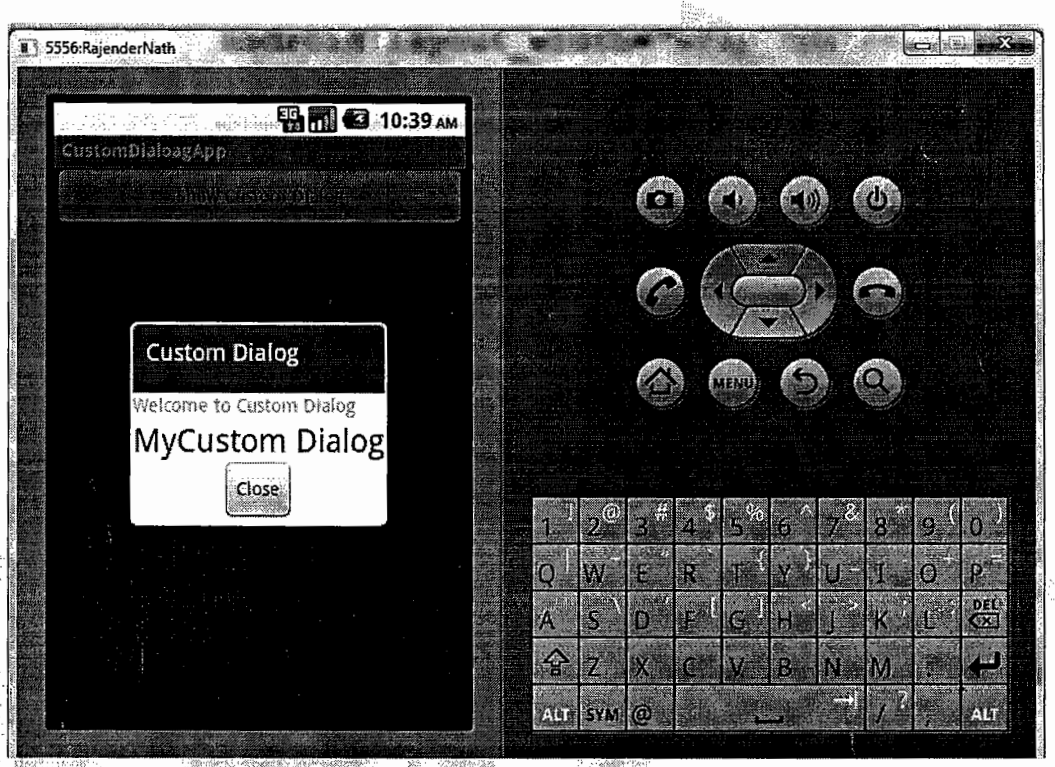
CustomizeDialog.java

```
package com.rajender.customdialog;

import android.app.Activity;
import android.os.Bundle;
import android.view.View;

 public class MyActivity extends Activity {
   @Override
   public void onCreate(Bundle b) {
     super.onCreate(b);
     setContentView(R.layout.main);
   }
   public void showCustomDialog(View v){
        MyCustomDialog dialog=new MyCustomDialog(this);
              dialog.setTitle("Custom Dialog");
              dialog.show();
   }
}
```

MyCustomDialog.java

```
package com.rajender.customdialog;

import android.app.Dialog;
import android.content.Context;
import android.view.View;
import android.widget.Button;
import android.view.View.OnClickListener;;

public class MyCustomDialog extends Dialog
        implements OnClickListener{
        Button button;
        public MyCustomDialog(Context context) {
                super(context);
                setContentView(R.layout.customdialog);
                button=(Button)findViewById(R.id.button);
                button.setOnClickListener(this);
```

```
    }
    public void onClick(View v) {
        dismiss();
    }
}
```



main.xml

```
<?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"
   android:background="#FFF8DC" >

   <TextView
      android:layout_width="fill_parent"
      android:layout_height="wrap_content"
      android:text="Student Registration"
      android:textColor="#f00"
      android:textSize="30px"
      android:textStyle="bold" />

   <TableRow
      android:layout_width="match_parent"
      android:layout_height="wrap_content" >

      <TextView
         android:layout_width="120px"
         android:layout_height="wrap_content"
         android:text="UserName"
         android:textColor="#00f"
         android:textSize="25px" />

      <EditText
         android:id="@+id/usetname"
         android:layout_width="fill_parent"
         android:layout_height="wrap_content" />
```

```
</TableRow>


<TableRow
   android:layout_width="match_parent"
   android:layout_height="wrap_content" >

   <TextView
      android:layout_width="120px"
      android:layout_height="wrap_content"
      android:text="Password"
      android:textColor="#00f"
      android:textSize="25px" />

   <EditText
      android:id="@+id/password"
      android:inputType="textPassword"
      android:layout_width="fill_parent"
      android:layout_height="wrap_content" />
</TableRow>

<TableRow
   android:layout_width="match_parent"
   android:layout_height="wrap_content" >

   <TextView
      android:layout_width="120px"
      android:layout_height="wrap_content"
      android:text="DOB"
      android:textColor="#00f"
      android:textSize="25px" />

   <EditText
      android:id="@+id/dateOfBirth"
      android:layout_width="140px"
      android:layout_height="wrap_content" />

   <Button
      android:onClick="datePickerFunction"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="Date" />

</TableRow>

<TableRow
   android:layout_width="match_parent"
   android:layout_height="wrap_content" >

   <TextView
      android:layout_width="120px"
      android:layout_height="wrap_content"
      android:text="Phone"
```

```
        android:textColor="#00f"
        android:textSize="25px" />

    <EditText
        android:id="@+id/phone"
        android:numeric="integer"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content" />
  </TableRow>

  <TableRow
    android:layout_width="match_parent"
    android:layout_height="wrap_content" >

    <TextView
        android:layout_width="120px"
        android:layout_height="wrap_content"
        android:text="Address"
        android:textColor="#00f"
        android:textSize="25px" />

    <EditText
        android:id="@+id/address"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content" />
  </TableRow>

  <TableRow
    android:layout_width="match_parent"
    android:layout_height="wrap_content" >

     <Button
        android:onClick="submitFunction"
        android:layout_width="150px"
        android:layout_height="wrap_content"
        android:text="Submit" />

     <Button
        android:onClick="resetFunction"
        android:layout_width="150px"
        android:layout_height="wrap_content"
        android:text="Reset" />

  </TableRow>

</LinearLayout>
```

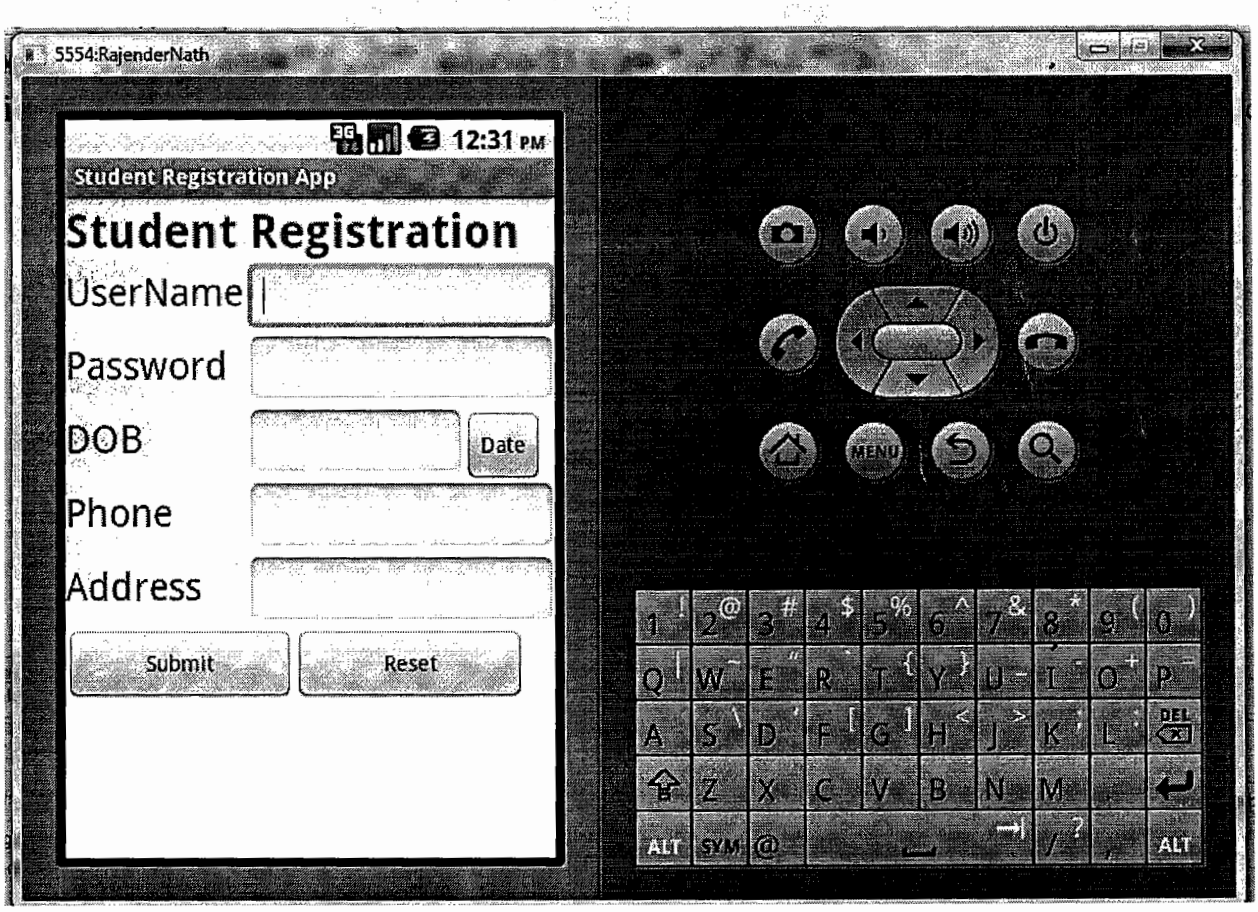
StudentRegistrationActivity.java

```
package com.rajender.student.dialog;
```

```
import java.util.Calendar;
import android.app.Activity;
import android.app.AlertDialog;
import android.app.DatePickerDialog;
import android.os.Bundle;
import android.view.View;
import android.widget.DatePicker;
import android.widget.EditText;
import android.widget.Toast;

public class StudentRegistrationActivity extends Activity{
   EditText username,password,dateOfBirth,phone,address;
        String user,pass,dob,ph,addr;
        AlertDialog.Builder dialog;

   @Override
   public void onCreate(Bundle bundle) {
      super.onCreate(bundle);
      setContentView(R.layout.main);

      dialog=new AlertDialog.Builder(this);
      dialog.setNeutralButton("OK",null);

      username=(EditText)findViewById(R.id.usetname);
      password=(EditText)findViewById(R.id.password);
      dateOfBirth=(EditText)findViewById(R.id.dateOfBirth);
      phone=(EditText)findViewById(R.id.phone);
      address=(EditText)findViewById (R.id.address);
   }

  //Submit Function
   public void submitFunction(View v){
        //Get the EditText values and store in a String
        user=username.getText().toString().trim();
        pass=password.getText().toString().trim();
        dob=dateOfBirth.getText().toString().trim();
        ph=phone.getText().toString().trim();
        addr=address.getText().toString().trim();

        //Validate EditText fields
        if(user.equals("") && pass.equals("") &&
                        dob.equals("") && ph.equals("")
                        && addr.equals("")){

                dialog.setMessage("Enter All Values");
                dialog.show();
        }

        else if(user.equals("")){
                dialog.setMessage("Enter Username");
                dialog.show();
        }

        else if(pass.equals("")){
```

```
        else if(pass.equals("")){
                dialog.setMessage("Enter Password");
                dialog.show();
        }

        else if(dob.equals("")){
                dialog.setMessage("Enter Date of Birth");
                dialog.show();
        }

        else if(ph.equals("")){
                dialog.setMessage("Enter Phone Number");
                dialog.show();
        }

        else if(addr.equals("")){
                dialog.setMessage("Enter Address");
                dialog.show();
        }

        else if(user.equals(pass)){
                dialog.setMessage(
                        "Username and Password " +
                        "should not be same");
                dialog.show();
        }
        else{

                Toast.makeText(getApplicationContext(),
                                "UserName : "+user+"\n"+
                                "Password : "+pass+"\n"+
                                "DateOfBirth : "+dob+"\n"+
                                "Phone    : "+ph+"\n"+
                                "Address : "+addr,
                                Toast.LENGTH_LONG).show();
                }
        }

public void resetFunction(View v){
                //Reset Values
                username.setText("");
                password.setText("");
                dateOfBirth.setText("");
                phone.setText("");
                address.setText("");
}


public void datePickerFunction(View v){
                //Get the System date
                Calendar c=Calendar.getInstance();
                        int cyear=c.get(Calendar.YEAR);
                        int cmonth=c.get(Calendar.MONTH);
```

```
        //create DatePickerDialog object
        DatePickerDialog dialog=new DatePickerDialog(this,
                                                dateList, cyear,cmonth,cday);
                        dialog.show();
                        }
        //Create Listener object and handle event
        DatePickerDialog.OnDateSetListener dateList
                        =new DatePickerDialog.OnDateSetListener(){
                @Override
                public void onDateSet(DatePicker view,
                                                        int year, int month,int day) {
                //Provide Business logic
                String date=month+1+"-"+day+"-"+year;
                        dateOfBirth.setText(date);

                }
        };
}
```



Creating Menus

Menus are an important part of any application. They provide familiar interfaces that reveal application functions and settings. Android offers an easy programming interface for developers to provide standardized application menus for various situations.

Android offers three fundamental types of application menus:

Options Menu

This is the primary set of menu items for an Activity. It is revealed by pressing the device MENU key. Within the Options Menu are two groups of menu items:

Icon Menu

This is the collection of items initially visible at the bottom of the screen at the press of the MENU key. It supports a maximum of six menu items. These are the only menu items that support icons and the only menu items that do not support checkboxes or radio buttons.

Expanded Menu

This is a vertical list of items exposed by the "More" menu item from the Icón Menu. It exists only when the Icon Menu becomes over-loaded and is comprised of the sixth Option Menu item and the rest.

Context Menu

This is a floating list of menu items that may appear when you perform a long-press on a View (such as a list item).

Submenu

This is a floating list of menu items that is revealed by an item in the Options Menu or a Context Menu. A Submenu item cannot support nested Submenus.

The add() method used in this sample takes four arguments: groupId, itemId, order, and title. The groupId allows you to associate this menu item with a group of other items

Options Menu

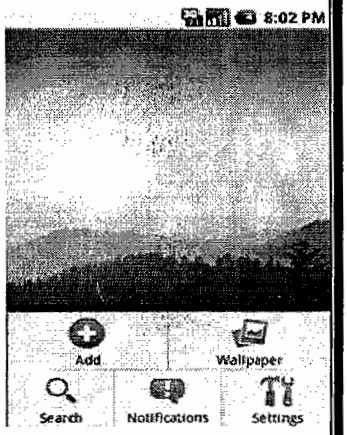
The Options Menu is opened by pressing the device MENU key. When opened, the Icon Menu is displayed, which holds the first six menu items. If more than six items are added to the Options Menu, then those that can't fit in the Icon Menu are revealed in the Expanded Menu, via the "More" menu item. The Expanded Menu is automatically added when there are more than six items.

The Options Menu is where you should include basic application functions and any necessary navigation items (e.g., to a home screen or application settings). You can also add Submenus for organizing topics and including extra menu functionality.

When this menu is opened for the first time, the Android system will call the Activity onCreateOptionsMenu() callback method. Override this method in your Activity and populate the Menu object given to you. You can populate the menu by inflating a menu resource that was defined in XML, or by calling add() for each item you'd like in the menu. This method adds a MenuItem, and returns the newly created object to you. You can use the returned MenuItem to set additional properties like an icon, a keyboard shortcut, an intent, and other settings for the item.

There are multiple add() methods. Usually, you'll want to use one that accepts an itemId argument. This is a unique integer that allows you to identify the item during a callback.

When a menu item is selected from the Options Menu, you will recieve a callback to the onOptionsItemSelected() method of your Activity. This callback passes you the MenuItem that has been selected. You can identify the item by requesting the itemId, with getItemId(), which returns the integer that was assigned with the add() method. Once you identify the menu item, you can take the appropriate action.



Here's an example of this procedure, inside an Activity, wherein we create an Options Menu and handle item selections:

```
/* Creates the menu items */
public boolean onCreateOptionsMenu(Menu menu) {
    menu.add(0, MENU_NEW_GAME, 0, "New Game");
    menu.add(0, MENU_QUIT, 0, "Quit");
    return true;
}

/* Handles item selections */
public boolean onOptionsItemSelected(MenuItem item) {
    switch (item.getItemId()) {
    case MENU_NEW_GAME:
        newGame();
        return true;
    case MENU_QUIT:
        quit();
        return true;
    }
    return false;
}
```

The add() method used in this sample takes four arguments: groupId, itemId, order, and title. The groupId allows you to associate this menu item with a group of other items (more about Menu groups) — in this example, we ignore it. itemId is a unique integer that we give the MenuItem so that can identify it in the next callback. order allows us to define the display order of the item — by default, they are displayed by the order in which we add them. title is, of course, the name that goes on the menu item (this can also be a string resource, and we recommend you do it that way for easier localization).

Tip: If you have several menu items that can be grouped together with a title, consider organizing them into a Submenu.

Adding icons

Icons can also be added to items that appears in the Icon Menu with setIcon().

For example:

```
menu.add(0, MENU_QUIT, 0, "Quit")
   .setIcon(R.drawable.menu_quit_icon);
```

Modifying the menu

If you want to sometimes re-write the Options Menu as it is opened, override the onPrepareOptionsMenu() method, which is called each time the menu is opened. This will pass you the Menu object, just like the onCreateOptionsMenu() callback. This is useful if you'd like to add or remove menu options depending on the current state of an application or game.

Note: When changing items in the menu, it's bad practice to do so based on the currently selected item. Keep in mind that, when in touch mode, there will not be a selected (or focused) item. Instead, you should use a Context Menu for such behaviors, when you want to provide functionality based on a particular item in the UI.

Context Menu

The Android context menu is similar, in concept, to the menu revealed with a "right-click" on a PC. When a view is registered to a context menu, performing a "long-press" (press and hold for about two seconds) on the object will reveal a floating menu that provides functions relating to that item. Context menus can be registered to any View object, however, they are most often used for items in a ListView, which helpfully indicates the presence of the context menu by transforming the background color of the ListView item when pressed. (The items in the phone's contact list offer an example of this feature.)

Note: Context menu items do not support icons or shortcut keys.

To create a context menu, you must override the Activity's context menu callback methods: onCreateContextMenu() and onContextItemSelected(). Inside the onCreateContextMenu()

callback method, you can add menu items using one of the add() methods, or by inflating a menu resource that was defined in XML. Then, register a ContextMenu for the View, with registerForContextMenu().

In onCreateContextMenu(), we are given not only the ContextMenu to which we will add MenuItems, but also the View that was selected and a ContextMenuInfo object, which provides additional information about the object that was selected. In this example, nothing special is done in onCreateContextMenu() — just a couple items are added as usual. In the onContextItemSelected() callback, we request the AdapterContextMenuInfo from the MenuItem, which provides information about the currently selected item. All we need from this is the list ID for the selected item, so whether editing a note or deleting it, we find the ID with the AdapterContextMenuInfo.info field of the object. This ID is passed to the editNote() and deleteNote() methods to perfrom the respective action.

Now, to register this context menu for all the items in a ListView, we pass the entire ListView to the registerForContextMenu(View) method:

Submenus

A sub menu can be added within any menu, except another sub menu. These are very useful when your application has a lot of functions that may be organized in topics, like the items in a PC application's menu bar (File, Edit, View, etc.). A sub menu is created by adding it to an existing Menu with addSubMenu(). This returns a SubMenu object (an extension of Menu). You can then add additional items to this menu, with the normal routine, using the add() methods. For example:

```
public boolean onCreateOptionsMenu(Menu menu) {
  boolean result = super.onCreateOptionsMenu(menu);

  SubMenu fileMenu = menu.addSubMenu("File");
  SubMenu editMenu = menu.addSubMenu("Edit");
  fileMenu.add("new");
  fileMenu.add("open");
  fileMenu.add("save");
  editMenu.add("undo");
  return result;
}
```

Callbacks for items selected in a sub menu are made to the parent menu's callback method. For the example above, selections in the sub menu will be handled by the onOptionsItemSelected() callback.

You can also add Submenus when you define the parent menu in XML.

Define Menus in XML

Just like Android UI layouts, you can define application menus in XML, then inflate them in your menu's onCreate...() callback method. This makes your application code cleaner and separates more interface design into XML, which is easier to visualize.

To start, create a new folder in your project res/ directory called menu. This is where you should keep all XML files that define your application menus.

In a menu XML layout, there are three valid elements: <menu>, <group> and <item>. The item and group elements must be children of a menu, but item elements may also be the children of a group, and another menu element may be the child of an item (to create a Submenu). Of course, the root node of any file must be a menu element.

As an example, we'll define the same menu created in the Options Menu section, above. We start with an XML file named options_menu.xml inside the res/menu/ folder:

```
<menu xmlns:android="http://schemas.android.com/apk/res/android">
   <item android:id="@+id/new_game"
         android:title="New Game" />
   <item android:id="@+id/quit"
         android:title="Quit" />
</menu>
```

Then, in the onCreateOptionsMenu() method, we inflate this resource using MenuInflater.inflate():

```
public boolean onCreateOptionsMenu(Menu menu) {
   MenuInflater inflater = getMenuInflater();
   inflater.inflate(R.menu.options_menu, menu);
   return true;
}
```

The getMenuInflater() method returns the MenuInflater for our activity's context. We then call inflate(), passing it a pointer to our menu resource and the Menu object given by the callback.

Menu groups

When adding new items to a menu, you can optionally include each item in a group. A menu group is a collection of menu items that can share certain traits, like whether they are visible, enabled, or checkable.

A group is defined by an integer (or a resource id, in XML). A menu item is added to the group when it is added to the menu, using one of the add() methods that accepts a groupId as an argument, such as add(int, int, int, int).

You can show or hide the entire group with setGroupVisible(); enable or disable the group with

setGroupEnabled(); and set whether the items can be checkable with setGroupCheckable().

Checkable menu items

Any menu item can be used as an interface for turning options on and off. This can be indicated with a checkbox for stand-alone options, or radio buttons for groups of mutually exlusive options (see the screenshot, to the right).

Note: Menu items in the Icon Menu cannot display a checkbox or radio button. If you choose to make items in the Icon Menu checkable, then you must personally indicate the state by swapping the icon and/or text each time the state changes between on and off.

To make a single item checkable, use the setCheckable() method, like so:

```
menu.add(0, VIBRATE_SETTING_ID, 0, "Vibrate")
   .setCheckable(true);
```

This will display a checkbox with the menu item (unless it's in the Icon Menu). When the item is selected, the onOptionsItemSelected() callback is called as usual. It is here that you must set the state of the checkbox. You can query the current state of the item with isChecked() and set the checked state with setChecked(). Here's what this looks like inside the onOptionsItemSelected() callback:

```
switch (item.getItemId()) {
case VIBRATE_SETTING_ID:
  if (item.isChecked()) item.setChecked(false);
  else item.setChecked(true);
  return true;
...
}
```

To make a group of mutually exclusive radio button items, simply assign the same group ID to each menu item and call setGroupCheckable(). In this case, you don't need to call setCheckable() on each menu items, because the group as a whole is set checkable. Here's an example of two mutually exclusive options in a Submenu:

```
SubMenu subMenu = menu.addSubMenu("Color");
subMenu.add(COLOR_MENU_GROUP, COLOR_RED_ID, 0, "Red");
subMenu.add(COLOR_MENU_GROUP, COLOR_BLUE_ID, 0, "Blue");
subMenu.setGroupCheckable(COLOR_MENU_GROUP, true, true);
```

In the setGroupCheckable() method, the first argument is the group ID that we want to set checkable. The second argument is whether we want the group items to be checkable. The last one is whether we want each item to be exclusively checkable (if we set this false, then all the

items will be checkboxes instead of radio buttons). When the group is set to be exclusive (radio buttons), each time a new item is selected, all other are automatically de-selected.

Shortcut keys

Quick access shortcut keys using letters and/or numbers can be added to menu items with setAlphabeticShortcut(char) (to set char shortcut), setNumericShortcut(int) (to set numeric shortcut), or setShortcut(char,int) (to set both). Case is not sensitive. For example:

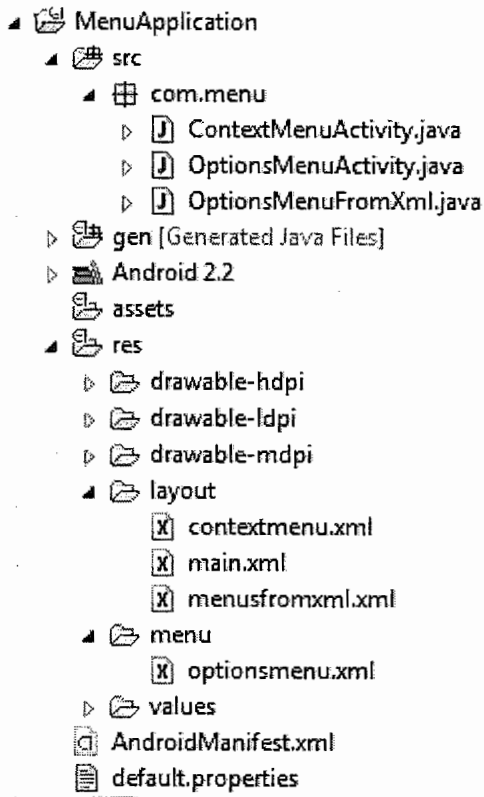
```
menu.add(0, MENU_QUIT, 0, "Quit")
   .setAlphabeticShortcut('q');
```

Now, when the menu is open (or while holding the MENU key), pressing the "q" key will select this item.

This shortcut key will be displayed as a tip in the menu item, below the menu item name (except for items in the Icon Menu).

Note: Shortcuts cannot be added to items in a Context Menu.

Menus Example :-



OptionsMenuActivity.java

```
package com.menu;

import android.app.Activity;
import android.content.Intent;
import android.os.Bundle;
import android.view.Menu;
import android.view.MenuItem;
import android.view.SubMenu;
import android.widget.Toast;

public class OptionsMenuActivity extends Activity {
   @Override
   public void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.main);
   }
```

```
@Override
public boolean onCreateOptionsMenu(Menu menu) {
        menu.add("Settings").setAlphabeticShortcut('S');
        menu.add("Dialpad");

    SubMenu sm1 = menu.addSubMenu("File");
        sm1.add("open");
        sm1.add("new");
        sm1.add("close");
        sm1.add("save");
        sm1.add("saveAs");
        sm1.add("exit");

    SubMenu sm2 = menu.addSubMenu("Edit");
        sm2.add("Copy");
        sm2.add("Cut");
        sm2.add("Paste");
        sm2.add("Undo");

        menu.add("Games");
        menu.add("Profiles");
        menu.add("Gallery").setIcon(R.drawable.icon);

    return true;
}

    @Override
    public boolean onOptionsItemSelected(MenuItem item) {
        if(item.getTitle()== "File"){
            Toast.makeText(getApplicationContext(),
                    "File Selected", Toast.LENGTH_LONG).show();
        }
        else if(item.getTitle()== "Games"){
            Toast.makeText(getApplicationContext(),
                    "Games Selected", Toast.LENGTH_LONG).show();
        }
```

```
            else if(item.getTitle()== "Dialpad"){
                    Intent in=new Intent(Intent.ACTION_DIAL);
                    startActivity(in);
            }
            else if(item.getTitle()== "Settings"){
                    Toast.makeText(getApplicationContext(),
                                    "Settings Selected", Toast.LENGTH_LONG).show();
            }
            return true;
        }
    }
```

Menus from Xml

optionsmenu.xml

```
<?xml version="1.0" encoding="utf-8"?>
<menu
 xmlns:android="http://schemas.android.com/apk/res/android">
      <item android:id="@+id/editId"
      android:title="Edit"
      android:icon="@drawable/icon" />
      <item android:id="@+id/fileId"
      android:title="File"
      android:icon="@drawable/icon" />
      <item android:id="@+id/pasteId"
      android:title="Paste"
      android:icon="@drawable/icon" />

</menu>
```

menusfromxml.xml

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

```
 xmlns:android="http://schemas.android.com/apk/res/android"
 android:orientation="vertical"
 android:layout_width="fill_parent"
 android:layout_height="fill_parent">

<TextView
        android:text="Click Menu Button"
        android:id="@+id/TextView01"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"/>
</LinearLayout>
```

OptionsMenuFromXml.java

```
package com.menu;

import android.app.Activity;
import android.os.Bundle;
import android.view.Menu;
import android.view.MenuInflater;

public class OptionsMenuFromXml extends Activity {
        @Override
        protected void onCreate(Bundle savedInstanceState) {
                super.onCreate(savedInstanceState);
                setContentView(R.layout.menusfromxml);
        }
        @Override
        public boolean onCreateOptionsMenu(Menu menu) {
                MenuInflater inf = new MenuInflater(this);
                inf.inflate(R.menu.optionsmenu, menu);
                return true;
        }
}
```

ContextMenu

ContextMenuActivity.java

```
package com.menu;
```

```
import android.app.Activity;
import android.content.Intent;
import android.os.Bundle;
import android.view.ContextMenu;
import android.view.MenuItem;
import android.view.SubMenu;
import android.view.View;
import android.view.ContextMenu.ContextMenuInfo;
import android.widget.Button;
import android.widget.Toast;

public class ContextMenuActivity extends Activity {
        Button button;
        @Override
        protected void onCreate(Bundle savedInstanceState) {
                super.onCreate(savedInstanceState);
                setContentView(R.layout.contextmenu);
                button=(Button)findViewById(R.id.contextButton);
                registerForContextMenu(button);
        }
        @Override
        public void onCreateContextMenu(ContextMenu menu, View v,
                        ContextMenuInfo menuInfo) {

                        menu.add("Settings");
                        menu.add("Dialpad");
                //adding submenus
                SubMenu sm1 = menu.addSubMenu("File");
                        sm1.add("open");
                        sm1.add("new");
                        sm1.add("close");
                        sm1.add("save");
                        sm1.add("saveAs");
                        menu.add("Games");
                        menu.add("Profiles");
```

```
    }
    @Override
    public boolean onContextItemSelected(MenuItem item) {
        if(item.getTitle()== "File"){
        Toast.makeText(getApplicationContext(),"File Selected",
        Toast.LENGTH_LONG).show();
        }
        else if(item.getTitle()== "Games"){
        Toast.makeText(getApplicationContext(),"Games Selected",
                    Toast.LENGTH_LONG).show();
        }
        else if(item.getTitle()== "Dialpad"){
            Intent in=new Intent(Intent.ACTION_DIAL);
            startActivity(in);
        }

        return true;
    }
}
```

main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
 xmlns:android="http://schemas.android.com/apk/res/android"
 android:layout_width="wrap_content"
 android:layout_height="wrap_content">
<Button
    android:text="Long Press"
    android:id="@+id/contextButton"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"/>
</LinearLayout>
```

Types Resource

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
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
Define a color resources that changes based on the View state.
Saved in res/color/ and accessed from the R.color class.
Drawable Resources
Define various graphics with bitmaps or XML.
Saved in res/drawable/ and accessed from the R.drawable class.
Layout Resource
Define the layout for your application UI.
Saved in res/layout/ and accessed from the R.layout class.
Menu Resource
Define the contents of your application menus.
Saved in res/menu/ and accessed from the R.menu class.
String Resources
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
Define the look and format for UI elements.
Saved in res/values/ and accessed from the R.style class.
More Resource Types
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.).

String Resources

A string resource provides text strings for your application with optional text styling and formatting. There are three types of resources that can provide your application with strings:

String
XML resource that provides a single string.
String Array
XML resource that provides an array of strings.

String

A single string that can be referenced from the application or from other resource files (such as an XML layout).

Note: A string is a simple resource that is referenced using the value provided in the name attribute (not the name of the XML file). So, you can combine string resources with other simple resources in the one XML file, under one <resources> element.

file location:

res/values/*filename*.xml

The filename is arbitrary. The <string> element's name will be used as the resource ID.

compiled resource datatype:

Resource pointer to a String.

resource reference:

In Java: R.string.*string_name*

In XML:@string/*string_name*

syntax:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
  <string
    name="string_name"
    >text_string</string>
</resources>
```

elements:

<resources>

Required. This must be the root node.

No attributes.

<string>

A string, which can include styling tags. Beware that you must escape apostrophes and quotation marks.

attributes:

name

String. A name for the string. This name will be used as the resource ID.

example:

XML file saved at res/values/strings.xml:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
  <string name="hello">Hello!</string>
</resources>
```

This layout XML applies a string to a View:

```
<TextView
  android:layout_width="fill_parent"
  android:layout_height="wrap_content"
  android:text="@string/hello" />
```

This application code retrieves a string:

String string = getString(R.string.hello);

You can use either getString(int) or getText(int) to retieve a string. getText(int) will retain any rich text styling applied to the string.

String Array

An array of strings that can be referenced from the application.

Note: A string array is a simple resource that is referenced using the value provided in the name attribute (not the name of the XML file). As such, you can combine string array resources with other simple resources in the one XML file, under one <resources> element.

file location:

res/values/*filename*.xml

The filename is arbitrary. The <string-array> element's name will be used as the resource ID.

compiled resource datatype:

Resource pointer to an array of Strings.

resource reference:

In Java: R.array.*string_array_name*

syntax:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string-array
        name="string_array_name">
        <item
            >text_string</item>
    </string-array>
</resources>
```

elements:

<resources>

Required. This must be the root node.

No attributes.

<string-array>

Defines an array of strings. Contains one or more <item> elements.

attributes:

name

String. A name for the array. This name will be used as the resource ID to reference the array.

<item>
A string, which can include styling tags. The value can be a referenced to another string resource. Must be a child of a <string-array> element. Beware that you must escape apostrophes and quotation marks.

example:

XML file saved at res/values/strings.xml:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
  <string-array name="week">
    <item>Sunday</item>
    <item>Monday</item>
    <item>Tuesday</item>
    <item>Wednesday</item>
    <item>Thursday</item>
    <item>Fridayday</item>
    <item>Saturday</item>
  </string-array>
</resources>
```

This application code retrieves a string array:

```
Resources res = getResources();
String[] weekDays = res.getStringArray(R.array.week);
```

Styling with HTML markup

You can add styling to your strings with HTML markup. For example:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
  <string name="welcome">Welcome to <b>Android</b>!</string>
</resources>
```

Supported HTML elements include:

- <b> for **bold** text.
- <i> for *italic* text.
- <u> for underline text.

Sometimes you may want to create a styled text resource that is also used as a format string. Normally, this won't work because the String.format(String, Object...) method will strip all the style information from the string. The work-around to this is to write the HTML tags with escaped entities, which are then recovered with fromHtml(String), after the formatting takes place. For example:

1. Store your styled text resource as an HTML-escaped string:

```
<resources>
  <string name="welcome_messages">Hello, %1$s! You have &lt;b>%2$d new
messages&lt;/b>.</string>
</resources>
```

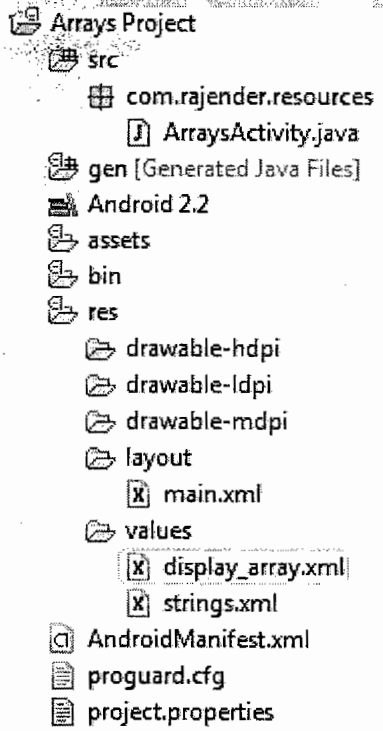
In this formatted string, a <b> element is added. Notice that the opening bracket is HTML-escaped, using the < notation.

2. Then format the string as usual, but also call fromHtml(String) to convert the HTML text into styled text:

```
Resources res = getResources();
String text = String.format(res.getString(R.string.welcome_messages), username,
mailCount);
CharSequence styledText = Html.fromHtml(text);
```

Because the fromHtml(String) method will format all HTML entities, be sure to escape any possible HTML characters in the strings you use with the formatted text, using htmlEncode(String). For instance, if you'll be passing a string argument to String.format() that may contain characters such as "<" or "&", then they must be escaped before formatting, so that when the formatted string is passed through fromHtml(String), the characters come out the way they were originally written. For example:

```
String escapedUsername = TextUtil.htmlEncode(username);
Resources res = getResources();
String text = String.format(res.getString(R.string.welcome_messages), escapedUsername,
mailCount);
CharSequence styledText = Html.fromHtml(text);
```



ArraysActivity.java

```
package com.rajender.resources;
import java.util.ArrayList;
import android.app.Activity;
import android.content.res.Resources;
import android.os.Bundle;
import android.widget.ArrayAdapter;
import android.widget.ListView;

public class ArraysActivity extends Activity {
   ListView strList,intList;
   ArrayList<Integer> al;
   @Override
   public void onCreate(Bundle b) {
      super.onCreate(b);
      setContentView(R.layout.main);
      al=new ArrayList<Integer>();

      Resources res=getResources();
      //Display String-Array
      String[] weekDays = res.getStringArray(R.array.week);
      ArrayAdapter<String> strAdapter=
                  new ArrayAdapter<String>(this,
                     android.R.layout.simple_list_item_1, weekDays);

      strList=(ListView)findViewById(R.id.strView);
      strList.setAdapter(strAdapter);

      //Display Integer-Array
      int[] numbers = res.getIntArray(R.array.num);
      //Read and convert into Integer
      for(int i=0;i<numbers.length;i++){
         al.add(numbers[i]);
      }

      ArrayAdapter<Integer> intAdapter = new ArrayAdapter<Integer>(this,
                  android.R.layout.simple_list_item_1, al);
                  intList=(ListView)findViewById(R.id.intView);
            intList.setAdapter(intAdapter);
     }
    }
```

main.xml

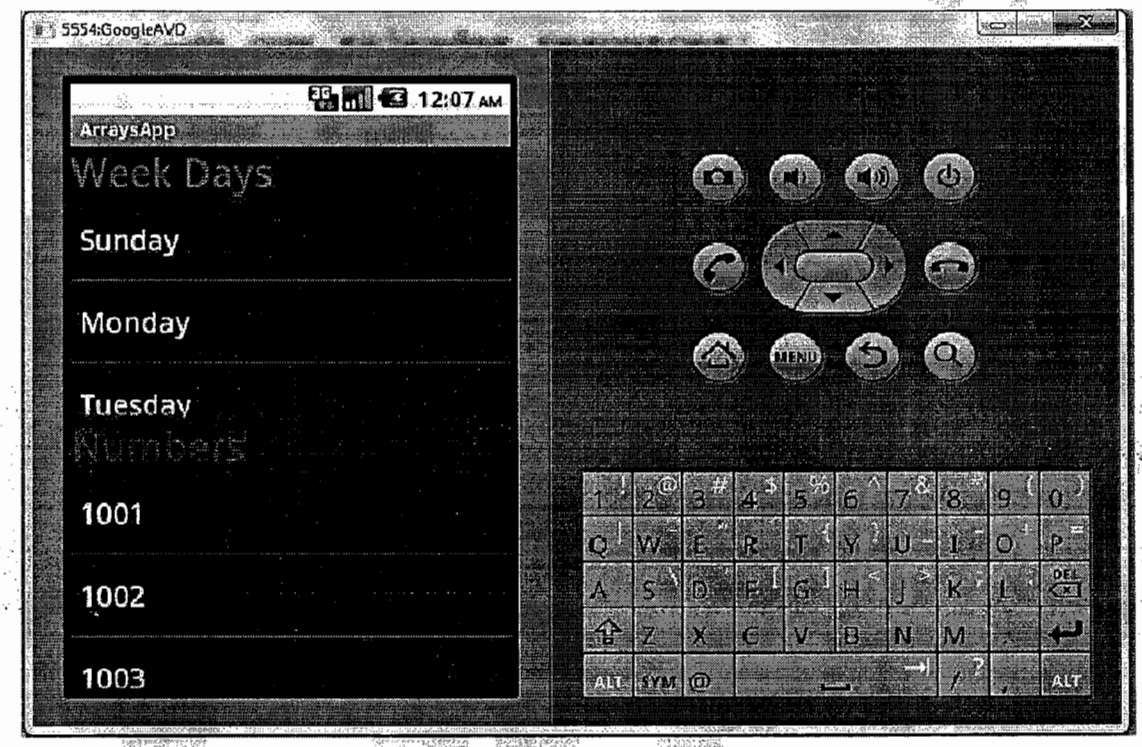
```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
   android:layout_width="fill_parent"
```

```
    android:layout_height="fill_parent"
    android:orientation="vertical" >

    <TextView
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="Week Days"
        android:textSize="30px"
        android:textColor="#f00"/>
    <ListView
        android:id="@+id/strView"
        android:layout_width="match_parent"
        android:layout_height="175px" >
    </ListView>

    <TextView
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="Numbers"
        android:textSize="30px"
        android:textColor="#00f"/>
    <ListView
        android:id="@+id/intView"
        android:layout_width="match_parent"
        android:layout_height="175px" >
    </ListView>

</LinearLayout>
```

display_array.xml

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string-array name="week">
        <item>Sunday</item>
        <item>Monday</item>
        <item>Tuesday</item>
        <item>Wednesday</item>
        <item>Thursday</item>
        <item>Friday</item>
        <item>Saturday</item>
    </string-array>

    <integer-array name="num">
        <item>1001</item>
        <item>1002</item>
        <item>1003</item>
```

```
        <item>1004</item>
        <item>1005</item>
        <item>1006</item>
        <item>1007</item>
        <item>1008</item>
        <item>1009</item>
        <item>1010</item>
    </integer-array>
</resources>
```



User Interface Components

Different Components of User Interfaces.

The basic components.

- Activity (Screen)
- Views (ViewGroup) or Controls or Widgets
- Layouts

Activity: - Activities represents the screen that is being displayed. Assigning a view to the Activity will displays the User Interfaces.

Views: - All the User interfaces are divided from views. Views are the base classes for all UI elements.

ViewGroups: - Multiple views can be combined together into ViewGroups.
ViewGroups are used to manage the layout of child views.
Views are two types.

1. Passing a view instance to the setContentView( ) method in the Activity.
2. Passing a layout resource id to the setContentView( ) method with the findViewById( ) method.

- Second method is best for creating UI. Because by mentioning the layout resource id separates the presentation layer from the application logic.
- Code reducing will be great level.
- Reusability of the code.
- Different layouts can be made fit to different mobile devices of various configurations.

  Note: - portrait mode/ landscape mode

Layouts

- Layouts are used to position, the control of the user interfaces.
- Layouts are defined the position of the child views and the layout structure.
- ViewGroup class serves as the base class for the layouts (android.text.Layout).

1. **FrameLayout**: - A FrameLayout add the controls to the Top left corner of the screen. Every new control to be added the stack with each new view occupying the old view. If we add new element then old will go to background and new will visible.
2. **LeanierLayout**:- It aligns the components either vertical or horizontal line. A vertical layout will be having a **column** of views. A horizontal layout will be having a **row** of views.
3. **RelativeLayout**: - It defines the positions of each child view relative to the other components and screen boundaries.
4. **TableLayout**: - A TableLayout allows displaying the components into rows and columns. Table row objects will be present in side table layout element.
   - Each table row objects defines a single row in the table. Each row as 0 or more cells. Each cell will be holding a single view object. The table will be having as

many cells as the row with the most number of cells. The width of a column is defined by the row with the widget cell in that column.

- A TableLayout can specify a column as shrinkable by calling setColumnShrinkable( ) method. If this is marked as shrinkable the column width can be reduced to fit the table in to its parent object.
- A column can be need stretching by calling setColumnStrechabel( ) method.
- A column can be hided by calling setColumnCollapsed( ) method.
- Cells should be added to a row in increasing column order, column numbers are index based (0).
- If a column number is not specified it will automatically increased to the next available column. If a column number is skipped it will be considered as empty cell in that row.

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 View Group (which inherits from View) so you can nest layouts. You could also create your own custom layout by making a class that inherits from View Group.
The standard Layouts are:

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

1. **Absolute Layout:**

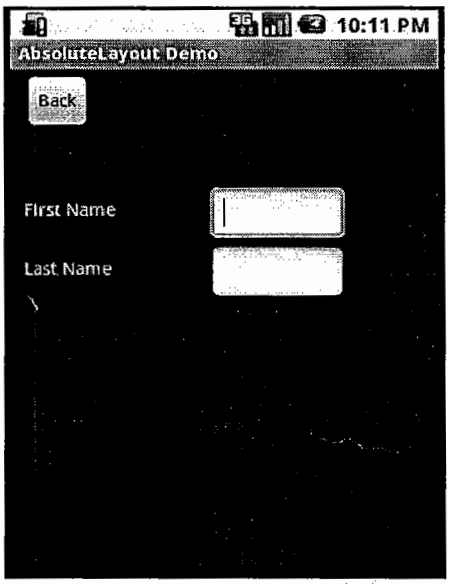
Absolute Layout is based on the simple idea of placing each control at an absolute position. You specify the exact x and y coordinates on the screen for each control. This is not recommended for most UI development (in fact Absolute Layout is currently deprecated) since absolutely positioning every element on the screen makes an inflexible UI that is much more difficult to maintain. Consider what happens if a control needs to be added to the UI. You would have to change the position of every single element that is shifted by the new control.

Here is a sample Layout XML using Absolute Layout.

```
<AbsoluteLayout xmlns:android="http://schemas.android.com/apk/res/android"
  android:layout_width="fill_parent"
  android:layout_height="fill_parent">
  <Button
        android:id="@+id/backbutton"
        android:text="Back"
```

```
        android:layout_x="10px"
        android:layout_y="5px"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" />
    <TextView
        android:layout_x="10px"
        android:layout_y="110px"
        android:text="First Name"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" />
    <EditText
        android:layout_x="150px"
        android:layout_y="100px"
        android:width="100px"
        android:layout_width="wrap content"
        android:layout_height="wrap content" />
    <TextView
        android:layout_x="10px"
        android:layout_y="160px"
        android:text="Last Name"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" />
        <EditText
        android:layout_x="150px"
        android:layout_y="150px"
        android:width="100px"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" />
</AbsoluteLayout>
```

Note how each element has android:layout_x and android:layout_y specified. Android defines the top left of the screen as (0,0) so the layout_x value will move the control to the right, and the layout_y value will move the control down. Here is a screenshot of the layout produced by this XML.



2. FrameLayout

FrameLayout is designed to display a single item at a time. You can have multiple elements within a FrameLayout but each element will be positioned based on the top left of the screen. Elements that overlap will be displayed overlapping. I have created a simple XML layout using FrameLayout that shows how this works.

```
<FrameLayout
        android:layout_width="fill_parent"
        android:layout_height="fill_parent"
        xmlns:android="http://schemas.android.com/apk/res/android">
        <ImageView
                android:src="@drawable/icon"
                android:scaleType="fitCenter"
                android:layout_height="fill_parent"
                android:layout_width="fill_parent"/>
        <TextView
                android:text="Learn-Android.com"
                android:textSize="24sp"
                android:textColor="#000000"
                android:layout_height="fill_parent"
                android:layout_width="fill_parent"
                android:gravity="center"/>
</FrameLayout>
```

Here is the result of this XML.

You can see I had both the ImageView and TextView fill the parent in both horizontal and vertical layout. Gravity specifies where the text appears within its container, so I set that to center. If I had not set a gravity then the text would have appeared at the top left of the screen.

FrameLayout can become more useful when elements are hidden and displayed programmatically. You can use the attribute android:visibility in the XML to hide specific elements. You can call setVisibility from the code to accomplish the same thing. The three available visibility values are visible, invisible (does not display, but still takes up space in the layout), and gone (does not display, and does not take space in the layout).

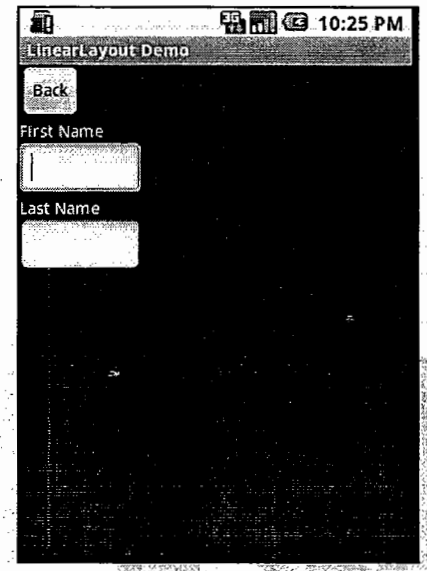
3.LinearLayout:

LinearLayout organizes elements along a single line. You specify whether that line is vertical or horizontal using android:orientation. Here is a sample Layout XML using LinearLayout.

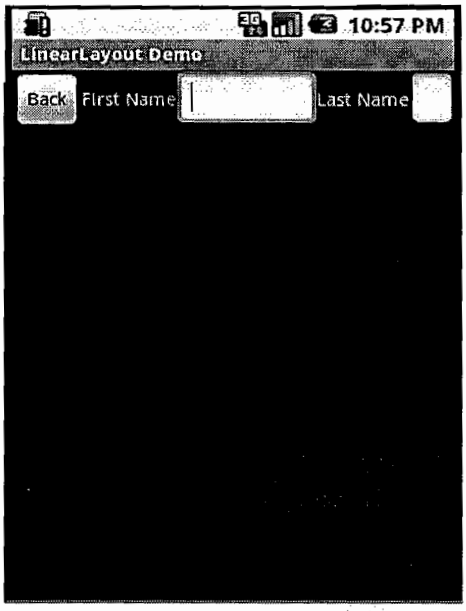
```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
        android:orientation="horizontal"
        android:layout_width="fill_parent"
    android:layout_height="fill_parent">
     <Button
        android:id="@+id/backbutton"
        android:text="Back"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" />
    <TextView
        android:text="First Name"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" />
    <EditText
        android:width="100px"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" />
```

```
    <TextView
            android:text="Last Name"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content" />
    <EditText
            android:width="100px"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content" />
</LinearLayout>
```

Here is a screenshot of the result of the above XML.



Here is a screenshot of the same XML except that the android:orientation has been changed to horizontal.



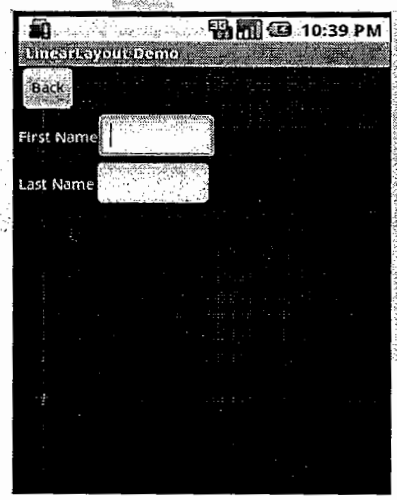
You might note that the EditText field at the end of the line has had its width reduced in order to fit. Android will try to make adjustments when necessary to fit items on screen. The last page of this tutorial will cover one method to help deal with this.

I mentioned on the first page that Layouts can be nested. LinearLayout is frequently nested, with horizontal and vertical layouts mixed. Here is an example of this.

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
        android:orientation="vertical"
        android:layout_width="fill_parent"
    android:layout_height="fill_parent">
     <Button
        android:id="@+id/backbutton"
        android:text="Back"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" />
    <LinearLayout
        android:orientation="horizontal"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content">
          <TextView
                android:text="First Name"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content" />
          <EditText
                android:width="100px"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content" />
    </LinearLayout>
```

```
<LinearLayout
        android:orientation="horizontal"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content">
          <TextView
                android:text="Last Name"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content" />
          <EditText
                android:width="100px"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content" />
  </LinearLayout>
</LinearLayout>
```

As you can see we have a Vertical LinearLayout whose children are a button, and two horizontal LinearLayouts. Each horizontal LinearLayout has two child controls. You should note that in the child LinearLayout elements I used android:layout_height="wrap_content" instead of fill_parent. If I had used fill_parent the first name TextView and EditView would have taken all of the available space on the screen, and the Last Name would not have been displayed. Here is what this XML does display.



Nested Layouts do not have to be of one type. I could, for example, have a LinearLayout as one of the children in a FrameLayout.

4.RelativeLayout:

RelativeLayout lays out elements based on their relationships with one another, and with the parent container. This is arguably the most complicated layout, and we need several properties to actually get the layout we want.

Relative To Container :

These properties will layout elements relative to the parent container.

- android:layout_alignParentBottom – Places the bottom of the element on the bottom of the container
- android:layout_alignParentLeft – Places the left of the element on the left side of the container
- android:layout_alignParentRight – Places the right of the element on the right side of the container
- android:layout_alignParentTop – Places the element at the top of the container
- android:layout_centerHorizontal – Centers the element horizontally within its parent container
- android:layout_centerInParent – Centers the element both horizontally and vertically within its container
- android:layout_centerVertical – Centers the element vertically within its parent container

Relative To Other Elements :

These properties allow you to layout elements relative to other elements on screen. The value for each of these elements is the id of the element you are using to layout the new element. Each element that is used in this way must have an ID defined using android:id="@+id/XXXXX" where XXXXX is replaced with the desired id. You use "@id/XXXXX" to reference an element by its id. One thing to remember is that referencing an element before it has been declared will produce an error.

- android:layout_above – Places the element above the specified element
- android:layout_below – Places the element below the specified element
- android:layout_toLeftOf – Places the element to the left of the specified element
- android:layout_toRightOf – Places the element to the right of the specified element

Alignment With Other Elements :

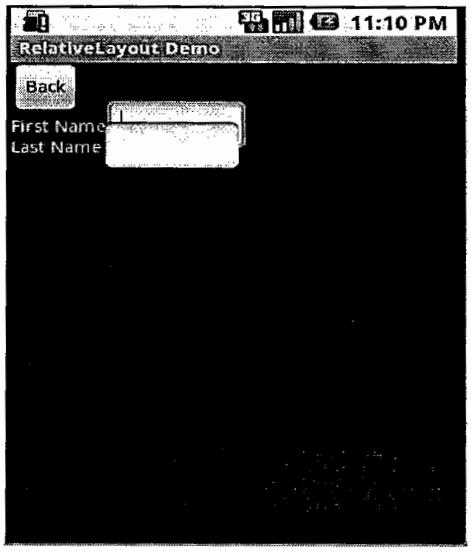
These properties allow you to specify how elements are aligned in relation to other elements.

- android:layout_alignBaseline – Aligns baseline of the new element with the baseline of the specified element
- android:layout_alignBottom – Aligns the bottom of new element in with the bottom of the specified element
- android:layout_alignLeft – Aligns left edge of the new element with the left edge of the specified element
- android:layout_alignRight – Aligns right edge of the new element with the right edge of the specified element
- android:layout_alignTop – Places top of the new element in alignment with the top of the specified element

Here is a sample XML Layout

```
<RelativeLayout
        android:layout_width="fill_parent"
        android:layout_height="fill_parent"
        xmlns:android="http://schemas.android.com/apk/res/android">
        <Button
                android:id="@+id/backbutton"
                android:text="Back"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content" />
        <TextView
                android:id="@+id/firstName"
                android:text="First Name"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:layout_below="@id/backbutton" />
        <EditText
                android:width="100px"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:layout_toRightOf="@id/firstName"
                android:layout_alignBaseline="@id/firstName" />
        <TextView
                android:id="@+id/lastName"
                android:text="Last Name"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:layout_below="@id/firstName" />
        <EditText
                android:width="100px"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:layout_toRightOf="@id/lastName"
                android:layout_alignBaseline="@id/lastName" />
</RelativeLayout>
```

Here is the screen produced by that XML.

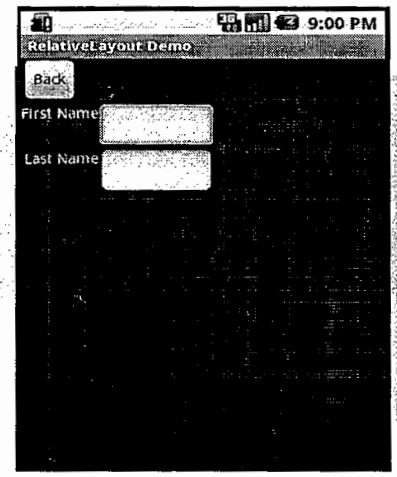


I wanted to show this to you because the first time I made a RelativeLayout I did exactly this and then looked at the screen and said, "Hang on a minute, that's not what I wanted!" The problem here is that when Android draws the TextView last Name below the TextView first Name it only sets aside the space it needs for the TextView. Android only reads the Layout XML one time so it doesn't know that an EditView is the next item and doesn't plan for it. So when the EditView is drawn to the right of the TextView it only has the height of the TextView to work with so it overlaps the EditView above it. Here is the Layout XML I wrote to create the form the way it should look.

```
<RelativeLayout
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    xmlns:android="http://schemas.android.com/apk/res/android">
    <Button
        android:id="@+id/backbutton"
        android:text="Back"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" />
    <TextView
        android:id="@+id/firstName"
        android:text="First Name"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_below="@id/backbutton" />
    <EditText
        android:id="@+id/editFirstName"
        android:width="100px"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_toRightOf="@id/firstName"
        android:layout_below="@id/backbutton"/>
```

```
<EditText
        android:id="@+id/editLastName"
        android:width="100px"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_below="@id/editFirstName"
        android:layout_alignLeft="@id/editFirstName"/>
<TextView
        android:id="@+id/lastName"
        android:text="Last Name"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_toLeftOf="@id/editLastName"
        android:layout_below="@id/editFirstName" />
</RelativeLayout>
```

You probably noticed that I had to rearrange the elements in the XML since, as I already mentioned, you cannot reference an element that has not already been laid out. Here is what the updated RelativeLayout produces.



5. TableLayout :

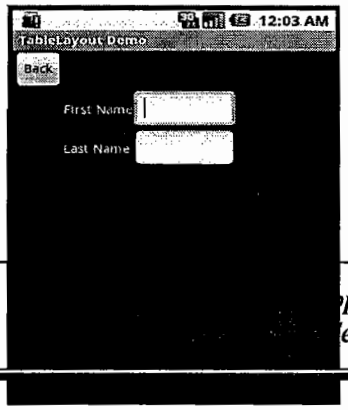
Table Layout organizes content into rows and columns. The rows are defined in the layout XML, and the columns are determined automatically by Android. This is done by creating at least one column for each element. So, for example, if you had a row with two elements and a row with five elements then you would have a layout with two rows and five columns.

You can specify that an element should occupy more than one column using android:layout_span. This can increase the total column count as well, so if we have a row with two elements and each element has android:layout_span="3" then you will have at least six columns in your table.

By default, Android places each element in the first unused column in the row. You can, however, specify the column an element should occupy using android:layout_column.

Here is some sample XML using Table Layout.

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

<TableRow>
                <Button
                android:id="@+id/backbutton"
                android:text="Back"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content" />
        </TableRow>
        <TableRow>
                <TextView
                android:text="First Name"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:layout_column="1" />
                <EditText
                android:width="100px"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content" />
        </TableRow>
        <TableRow>
                <TextView
                android:text="Last Name"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:layout_column="1" />
                <EditText
                android:width="100px"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content" />
        </TableRow>
</TableLayout>
```

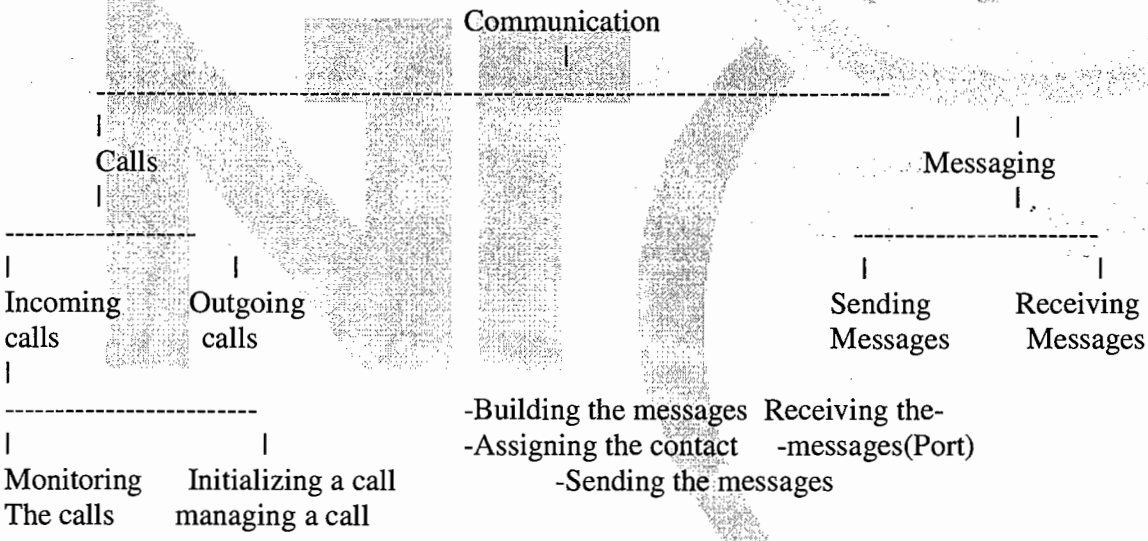


Telephony

- We are using mobiles for making a calls, sms.
- It is used to communication.

Communication is of two types.

1. Making Calls
2. Making Messages



Incoming messages:-

Normal Messages

Flash messages (Call summary)

SMS -- > Text Messages

MMS -- > Video& Audio Messages

Receiving messages by using PORT. We can monitor and it sends to **inbox.** We are not change PORT numbers.

Way of making the calls: -

1. Creating your Intent
2. <uses-permission>

1st way

Creating your Intent

Intent *myIntent*=new **Intent** (Intent.ACTION_DIAL, Uri.parseInt("9959538564"));
startActivity (*myIntent*);
or
startActivityForResult (Intent obj, Request Code);

2nd way

<uses-permission name= "android.permission.CALL_PHONE">
Telephony Manager will manage all these activities.
Phones are 2 types.
GSM(Any) --- > Global System for Mobile. ---- IMEI
CDMA (Specific) ----- >Code Division Multiple Access ---- MEID
IMEI --- > International Mobile Equipment Identity
MEID --- > Mobile Equipment Identifier.

TelephoneManager

- State of the Mobile
- Sim card (subscriber identity module)
- Uniqueness features (IMEI Number)
- Devices
- Phone type

Calls depends on network operator
Data --- >Data Activity
Data State

Points:-

TelephonyManager:-

- The TelephonyManager provides access to the information about the Telephony services on the devices. Applications will be using the methods in the TelephonyManager class to determine Telephone state, services, subscriber information. It consists of the android.telephony and android.telephony.gsm packages.
- Applications will be registering a listener to receive notifications of changes in the telephone state.
- We can create TelephonyManager object by using getSystemService() method as follows.

TelephonyManager tm=**(TelepnonyManager)**

Constants

String	ACTION_PHONE_STATE_CHANGED	Broadcast intent action indicating that the call state (cellular) on the device has changed.
int	CALL_STATE_IDLE	Device call state: No activity.
int	CALL_STATE_OFFHOOK	Device call state: Off-hook.
int	CALL_STATE_RINGING	Device call state: Ringing.
int	DATA_ACTIVITY_DORMANT	Data connection is active, but physical link is down
int	DATA_ACTIVITY_IN	Data connection activity: Currently receiving IP PPP traffic.
int	DATA_ACTIVITY_INOUT	Data connection activity: Currently both sending and receiving IP PPP traffic.
int	DATA_ACTIVITY_NONE	Data connection activity: No traffic.
int	DATA_ACTIVITY_OUT	Data connection activity: Currently sending IP PPP traffic.
int	DATA_CONNECTED	Data connection state: Connected.
int	DATA_CONNECTING	Data connection state: Currently setting up a data connection.
int	DATA_DISCONNECTED	Data connection state: Disconnected.
int	DATA_SUSPENDED	Data connection state: Suspended.
String	EXTRA_INCOMING_NUMBER	The lookup key used with the ACTION_PHONE_STATE_CHANGED broadcast for a String containing the incoming phone number.
String	EXTRA_STATE	The lookup key used with the ACTION_PHONE_STATE_CHANGED broadcast for a String containing the new call state.
int	NETWORK_TYPE_CDMA	Current network is CDMA: Either IS95A or IS95B
int	NETWORK_TYPE_EDGE	Current network is EDGE
int	NETWORK_TYPE_EHRPD	Current network is eHRPD
int	NETWORK_TYPE_GPRS	Current network is GPRS
int	NETWORK_TYPE_HSDPA	Current network is HSDPA
int	NETWORK_TYPE_HSPA	Current network is HSPA
int	NETWORK_TYPE_HSUPA	Current network is HSUPA

int	NETWORK_TYPE_IDEN	Current network is iDen
int	NETWORK_TYPE_LTE	Current network is LTE
int	NETWORK_TYPE_UMTS	Current network is UMTS
int	PHONE_TYPE_CDMA	Phone radio is CDMA.
int	PHONE_TYPE_GSM	Phone radio is GSM.
int	PHONE_TYPE_NONE	No phone radio.
int	PHONE_TYPE_SIP	Phone is via SIP.
int	SIM_STATE_ABSENT	SIM card state: no SIM card is available in the device
int	SIM_STATE_NETWORK_LOCKED	SIM card state: Locked: requries a network PIN to unlock
int	SIM_STATE_PIN_REQUIRED	SIM card state: Locked: requires the user's SIM PIN to unlock
int	SIM_STATE_PUK_REQUIRED	SIM card state: Locked: requires the user's SIM PUK to unlock
int	SIM_STATE_READY	SIM card state: Ready
int	SIM_STATE_UNKNOWN	SIM card state: Unknown.

Methods in TelephonyManager

Public Methods

- public int **getCallState** ()

  Returns a constant indicating the call state (cellular) on the device.

- public CellLocation **getCellLocation** ()

  Returns the current location of the device. Return null if current location is not available.

  Requires ermission: ACCESS_COARSE_LOCATION or ACCESS_FINE_LOCATION.

- public int **getDataActivity** ()

  Returns a constant indicating the type of activity on a data connection (cellular).

  - DATA_ACTIVITY_NONE
  - DATA_ACTIVITY_IN
  - DATA_ACTIVITY_OUT
  - DATA_ACTIVITY_INOUT

- public int **getDataState** ()

Returns a constant indicating the current data connection state (cellular).

- DATA_DISCONNECTED
- DATA_CONNECTING
- DATA_CONNECTED
- DATA_SUSPENDED

➢ public String **getDeviceId** ()

Returns the unique device ID, for example, the IMEI for GSM and the MEID or ESN for CDMA phones. Return null if device ID is not available.

Requires Permission: READ_PHONE_STATE

➢ public String **getDeviceSoftwareVersion** ()

Returns the software version number for the device, for example, the IMEI/SV for GSM phones. Return null if the software version is not available.

Requires Permission: READ_PHONE_STATE

➢ public String **getLine1Number** ()

Returns the phone number string for line 1, for example, the MSISDN for a GSM phone. Return null if it is unavailable.

Requires Permission: READ_PHONE_STATE

➢ public String **getNetworkCountryIso** ()

Returns the ISO country code equivalent of the current registered operator's MCC (Mobile Country Code).

Availability: Only when user is registered to a network. Result may be unreliable on CDMA networks (use getPhoneType()to determine if on a CDMA network).

➢ public String **getNetworkOperator** ()

Returns the numeric name (MCC+MNC) of current registered operator.

Availability: Only when user is registered to a network. Result may be unreliable on CDMA networks (use getPhoneType()to determine if on a CDMA network).

➢ public String **getNetworkOperatorName** ()

Returns the alphabetic name of current registered operator.

Availability: Only when user is registered to a network. Result may be unreliable on CDMA networks (use getPhoneType()to determine if on a CDMA network).

- public int **getNetworkType** ()

  Returns a constant indicating the radio technology (network type) currently in use on the device for data transmission.

Returns network type

- NETWORK_TYPE_UNKNOWN
- NETWORK_TYPE_GPRS
- NETWORK_TYPE_EDGE
- NETWORK_TYPE_UMTS
- NETWORK_TYPE_CDMA

- public int **getPhoneType** ()

Returns a constant indicating the device phone type. This indicates the type of radio used to transmit voice calls.

- PHONE_TYPE_NONE
- PHONE_TYPE_GSM
- PHONE_TYPE_CDMA
- PHONE_TYPE_SIP

- public String **getSimCountryIso** ()

  Returns the ISO country code equivalent for the SIM provider's country code.

- public String **getSimOperator** ()

  Returns the MCC+MNC (mobile country code + mobile network code) of the provider of the SIM. 5 or 6 decimal digits.

  Availability: SIM state must be SIM_STATE_READY

- public String **getSimOperatorName** ()

  Returns the Service Provider Name (SPN).

  Availability: SIM state must be SIM_STATE_READY

- public String **getSimSerialNumber** ()

  Returns the serial number of the SIM, if applicable. Return null if it is unavailable.

  Requires Permission: READ_PHONE_STATE

- public int **getSimState** ()

  Returns a constant indicating the state of the device SIM card.

- SIM_STATE_UNKNOWN
- SIM_STATE_ABSENT
- SIM_STATE_PIN_REQUIRED
- SIM_STATE_NETWORK_LOCKED
- SIM_STATE_READY

➢ public String **getSubscriberId** ()

Returns the unique subscriber ID, for example, the IMSI for a GSM phone. Return null if it is unavailable.

Requires Permission: READ_PHONE_STATE

➢ public String **getVoiceMailAlphaTag** ()

Retrieves the alphabetic identifier associated with the voice mail number.

Requires Permission: READ_PHONE_STATE

➢ public String **getVoiceMailNumber** ()

Returns the voice mail number. Return null if it is unavailable.

Requires Permission: READ_PHONE_STATE

➢ public boolean **hasIccCard** ()

Returns **true,** if ICC card is present

➢ public boolean **isNetworkRoaming** ()

Returns true if the device is considered roaming on the current network, for GSM purposes.

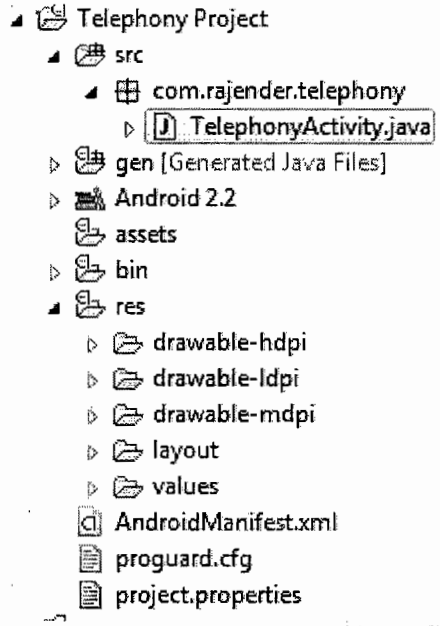
Availability: Only when user registered to a network.

➢ public void **listen** (PhoneStateListener listener, int events)

Registers a listener object to receive notification of changes in specified telephony states.

To register a listener, pass a PhoneStateListener and specify at least one telephony state of interest in the events argument. At registration, and when a specified telephony state changes, the telephony manager invokes the appropriate callback method on the listener object and passes the current (udpated) values.

To unregister a listener, pass the listener object and set the events argument to LISTEN_NONE (0).




```
package com.rajender.telephony;

import android.app.Activity;
import android.content.Context;
import android.os.Bundle;
import android.telephony.TelephonyManager;
import android.widget.Toast;

public class TelephonyActivity extends Activity {
   TelephonyManager tm;
        @Override
    public void onCreate(Bundle b) {
      super.onCreate(b);
      setContentView(R.layout.main);

      tm=(TelephonyManager)getSystemService
                                    (Context.TELEPHONY_SERVICE);

      String deviceId = tm.getDeviceId();
        Toast.makeText(getApplicationContext(), "Device Id is :  "+deviceId,
                                                  Toast.LENGTH_LONG).show();

         String swVersion = tm.getDeviceSoftwareVersion();
          Toast.makeText(getApplicationContext(),
                          "DeviceSoftwareVersion :  "+swVersion,
                                Toast.LENGTH_LONG).show();

         boolean roaming=tm.isNetworkRoaming();
          Toast.makeText(getApplicationContext(),
                      "NetworkRoaming :  "+roaming,
```

```
                                Toast.LENGTH_LONG).show();

    String nwCountryIso=tm.getNetworkCountryIso();
      Toast.makeText(getApplicationContext(),
              "NetworkCountryIso :  "+nwCountryIso,
               Toast.LENGTH_LONG).show();

    String simCountryIso=tm.getSimCountryIso();
      Toast.makeText(getApplicationContext(),
          "SimCountryIso :  "+simCountryIso, Toast.LENGTH_LONG).show();

int callState=tm.getCallState();

switch (callState) {
        case TelephonyManager.CALL_STATE_IDLE:
                Toast.makeText(getApplicationContext(),
                  "CALL_STATE_IDLE",  Toast.LENGTH_LONG).show();
                break;

        case TelephonyManager.CALL_STATE_OFFHOOK:
                Toast.makeText(getApplicationContext(),
                                "CALL_STATE_OFFHOOK",
                                Toast.LENGTH_LONG).show();
                break;
        case TelephonyManager.CALL_STATE_RINGING:
                Toast.makeText(getApplicationContext(),
                                "CALL_STATE_RINGING",
                                Toast.LENGTH_LONG).show();
                break;
        }
int phoneType=tm.getPhoneType();

switch (phoneType) {
        case TelephonyManager.PHONE_TYPE_CDMA:
                Toast.makeText(getApplicationContext(),
                                "PHONE_TYPE_CDMA",
                                Toast.LENGTH_LONG).show();
                break;

        case TelephonyManager.PHONE_TYPE_GSM:
                Toast.makeText(getApplicationContext(),
                                "PHONE_TYPE_GSM",
                                Toast.LENGTH_LONG).show();

                break;
```

```
                case TelephonyManager.PHONE_TYPE_NONE:
                        Toast.makeText(getApplicationContext(),
                                        "PHONE_TYPE_NONE",
                                        Toast.LENGTH_LONG).show();

                        break;
                }

        int simState=tm.getSimState();

        switch (simState) {
                case TelephonyManager.SIM_STATE_ABSENT:
                        Toast.makeText(getApplicationContext(),
                                        "SIM_STATE_ABSENT",
                                        Toast.LENGTH_LONG).show();
                        break;

                case TelephonyManager.SIM_STATE_READY:
                        Toast.makeText(getApplicationContext(),
                                        "SIM_STATE_READY",
                                        Toast.LENGTH_LONG).show();
                        break;
                case TelephonyManager.SIM_STATE_UNKNOWN:
                        Toast.makeText(getApplicationContext(),
                                        "SIM_STATE_UNKNOWN",
                                        Toast.LENGTH_LONG).show();

                        break;
                case TelephonyManager.SIM_STATE_NETWORK_LOCKED:
                        Toast.makeText(getApplicationContext(),
                                        "SIM_STATE_NETWORK_LOCKED",
                                        Toast.LENGTH_LONG).show();
                        break;
                }

    }
}
```

AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.rajender.telephony"
    android:versionCode="1"
    android:versionName="1.0" >

    <uses-sdk android:minSdkVersion="8" />
```

```
    <uses-permission android:name="android.permission.READ_PHONE_STATE"/>
  <application
    android:icon="@drawable/ic_launcher"
    android:label="@string/app_name" >
    <activity
      android:name="com.rajender.telephony.TelephonyActivity"
      android:label="@string/app_name" >
      <intent-filter>
        <action android:name="android.intent.action.MAIN" />

        <category android:name="android.intent.category.LAUNCHER" />
      </intent-filter>
    </activity>
  </application>

</manifest>
```

Android Location API and Google Maps

Android support Location based service APIs. Location service allows to find out the device current location. The application can ask for periodic update of the device location. The application can also register a intent receiver for proximity alerts like when the device is entering and existing from an area of given longitude, latitude and radius.

Let's check the important classes present in the android.location package.

Android Location API:

Following are the some important classes present under the android location package.

LocationManager: The class provides access to the location service. It also provides facility to get the best Location Provider as per the criteria. Proximity alerts can be set (as said above) with help of this class.

LocationProvider: It's an abstract superclass for location providers. A location provider provides periodic reports on the geographical location of the device.

LocationListener: Provides callback methods which are called when location gets changed. The listener object has to be registered with the location manager.

Criteria: The class provides the application to choose suitable Location Provider by providing access to set of required properties of the LocationProvider.

Android also provide an API to access the Google MAPs. So with the help of the Google MAPs and the location APIs the application can show required places to the user on the MAP.

Google Map API

Android defines a package called com.google.android.maps. The package contains classes related to rendering, controlling and overlaying information on the Google maps on the android devices. Let's see the most important classes defined in the package:

MapActivity: It is the spacing activity defined to show the Google MAPs. The MapActivity takes care of the low-level networking.

MapView: MapView is the view that supports and displays the map. This must be contained by a MapActivity.

MapController : MapController is the object used to move the map around the screen.

Overlay: It's a drawable object that can be shown on top of the map.

GeoPoint: It's a position in latitude-longitude.

Now we have some basic knowledge of the Location and Map APIs, so let's create some application and see them in action.

Application:

Let's develop an application that shown the Google MAP on the screen and shows the user's current position on the MAP. We will use Google MAP APIs to show map on the device and then use location APIs to get the device current location to show it on the MAP. The user location will gets updated if the user moved from the current location.

Application Activity:

To use map in an activity that activity has to be extended by the MapActivity as shown..

```
class MyGPSActivity extends MapActivity {
...
}
```

To use the Google MAP APIs, application AndroidManifest.xml file must define following XML element, as a child of the application element:

<uses-library android:name="com.google.android.maps" />

Using the MapView:

To display Map we need to add MapView to the application. Add following in the activity's layout file (main.xml).

```
<com.google.android.maps.MapView
android:id="@+id/myGMap"
android:layout_width="fill_parent"
android:layout_height="fill_parent"
android:enabled="true"
android:clickable="true"
android:apiKey="API_Key_String"
/>
```

To use the Google Map service an API key is needed. The API key can obtained as follows:

1) Get debug.keystore file. You will get this under USER_HOME\Local Settings\Application Data\Android directory.

2) Use keytool tool to generate Certificate fingerprint (MD5). Use following command on command prompt

keytool -list -alias androiddebugkey -keystore <path_to_debug_keystore>.keystore -storepass android -keypass android

3) Go to 'Sign Up for the Android Maps API' page. Put your Certificate fingerprint (MD5) And get your API key for android GMap application.

4) Replace "API_Key_String" with your API key.

Update the MyGPSActivity class to use the MapView

```
class MyGPSActivity extends MapActivity {
@Override
public void onCreate(Bundle savedInstanceState) {
// Creating and initializing Map
gMapView = (MapView) findViewById(R.id.myGMap);
GeoPoint p = new GeoPoint((int) (lat * 1000000), (int) (long * 1000000));
gMapView.setSatellite(true);
//get MapController that helps to set/get location, zoom etc.
mc = gMapView.getController();
mc.setCenter(p);
mc.setZoom(14);
}
...
}
```

Certain permission has to be set in the AndroidManifest.xml file to use location information.

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

<uses-permission
android:name="android.permission.ACCESS_COARSE_LOCATION"></uses-permission>

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

Using Location Manager:

The location manage object can be obtained with Context.getSystemService method with Context.LOCATION_SERVICE parameter.

LocationManager lm =
(LocationManager)getSystemService(Context.LOCATION_SERVICE);

Update the GPSActivity to implement the LocationListener interface so that the activity can listener to the location changes.

```
class MyGPSActivity extends MapActivity implements LocationListener {
...
/* This method is called when use position will get changed */
public void onLocationChanged(Location location) {
}
public void onProviderDisabled(String provider) {
}
public void onProviderEnabled(String provider) {
}
public void onStatusChanged(String provider, int status, Bundle extras) {
}
protected boolean isRouteDisplayed() {
return false;
}

}
```

Lets add code to initialize the LocationManager and register the Location Listener with the location manager in the onCreate() method

```
@Override

public void onCreate(Bundle savedInstanceState) {
```

```
...
LocationManager lm = (LocationManager)getSystemService(Context.LOCATION_SERVICE);
lm.requestLocationUpdates(LocationManager.GPS_PROVIDER, 1000L, 500.0f, this);


}
```

Now the onLocationChanged method of the GPSActivity method will be called if the user changes its position by 500m. A "gps" (GSP_PROVIDER) provider is used here but you can obtain a provider object as per your needs with the getBestProvider method of the LocationManger and the Criteria object.

Here is the implementation of the onLocationChanged method

```
public void onLocationChanged(Location location) {
if (location != null) {
double lat = location.getLatitude();
double lng = location.getLongitude();
p = new GeoPoint((int) lat * 1000000, (int) lng * 1000000);
mc.animateTo(p);
}
}
```

The code changes the map location to the new updated location.

We can add extra things in our application like Zoom controls, Marker and Text to show current location etc.

Adding Zoom control:

The MAP api provides facility to add zoom control to the map display. Following code add zoom control to your application:

```
// Adding zoom controls to Map
ZoomControls zoomControls = (ZoomControls) gMapView.getZoomControls();
zoomControls.setLayoutParams(new
ViewGroup.LayoutParams(LayoutParams.WRAP_CONTENT,
LayoutParams.WRAP_CONTENT));
```

```
gMapView.addView(zoomControls);

gMapView.displayZoomControls(true);
```

Adding Map Overlay:

A map overlay can be added showing the user's current location. To add an overlay, define a class that will extend Overlay class.

```
class MyLocationOverlay extends com.google.android.maps.Overlay {
@Override

public boolean draw(Canvas canvas, MapView mapView, boolean shadow, long when) {
super.draw(canvas, mapView, shadow);
Paint paint = new Paint();
// Converts lat/lng-Point to OUR coordinates on the screen.
Point myScreenCoords = new Point();
mapView.getProjection().toPixels(p, myScreenCoords);
paint.setStrokeWidth(1);
paint.setARGB(255, 255, 255, 255);
paint.setStyle(Paint.Style.STROKE);
Bitmap bmp = BitmapFactory.decodeResource(getResources(), R.drawable.marker);
canvas.drawBitmap(bmp, myScreenCoords.x, myScreenCoords.y, paint);
canvas.drawText("Here I am...", myScreenCoords.x, myScreenCoords.y, paint);
return true;

}

}
```

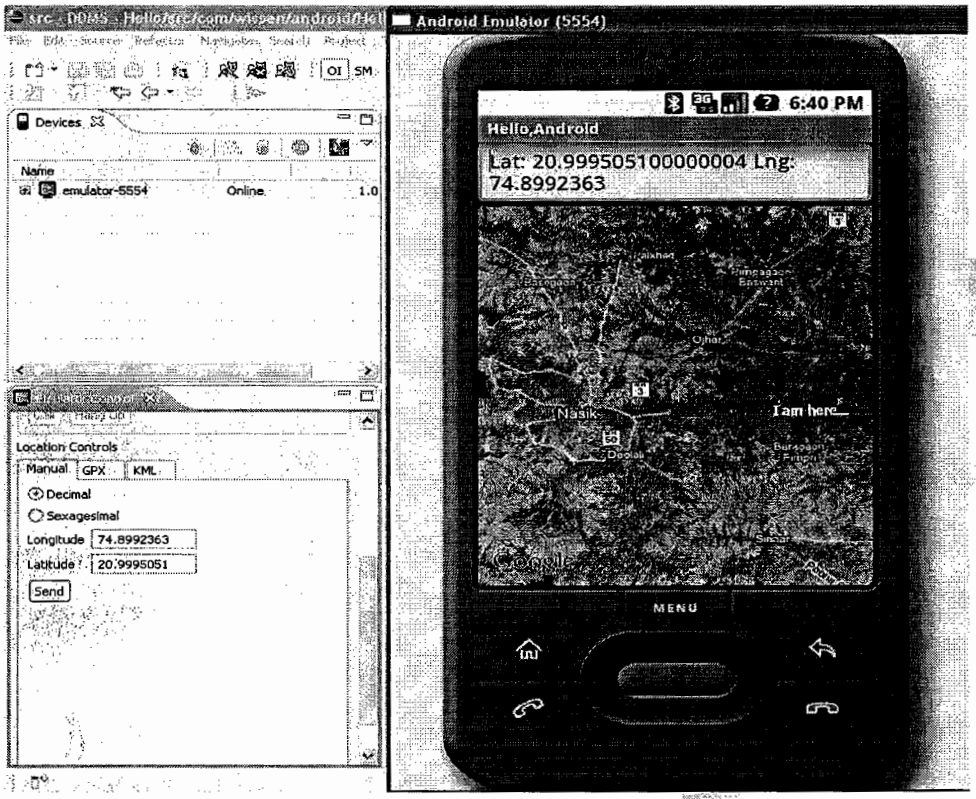
The overlay display a text "Here I am.." on the map at user location.Let's add this overlay to our map view

```
// Add a location mark
MyLocationOverlay myLocationOverlay = new MyLocationOverlay();
```

```
List<Overlay> list = gMapView.getOverlays();
list.add(myLocationOverlay);
```

Running application on Emulator:

Run the emulator instance. The latitude and longitude values can be supplied to the emulator with the help of 'Emulator Control' window present in the DDMS eclipse perspective. According to latitude and longitude application will display location of user.



Network Connectivity(WiFi)

Airtel
Vodofone
Tata
} Service providers/ wireless communications

Network connectivity

GPRS (General pocket radio service)

2G 2.5G 3.5G (3rd Generation)
(2nd Generation)

- It depends on (internet speed)
- Distance from the tower
- Signal strength generation N/W

Moves one place to another place the signals will be catch one n/w to another n/w (network carrier or mobile carrier).
ConnectivityManager: is the basic class to manage the network connectivity (it is similar to telephoney manager).
Once we get the ConnectivityManager, we can access the local connectivity i.e: WiFi

- Default access then we required ACESS_NETWORK_STATE is required
- While changing you network then CHANGE_NETWORK_STATE is required
- While downloading a file, if we click back button then if you want to confirm that task back ground then we use a settings.

Points: ConnectivityManager

- The **ConnectivityManager** gives the information about the state of network connectivity. It will show messages to the users whom the network connectivity changes.

- An instance of the connectivity manager is obtained by calling Context.getsSystemService (Context.CONNECTIVITY_SERVICE);

ConnectivityManager cm= (ConnectivityManager)getSystemService Context.CONNECTIVITY_SERVICE);

- The basic functionality of **ConnectivityManager** class are, it will monitoring the wifi(wireless fidivity)
- GPRS ⟶ General Packet Radio Service
- UMTS ⟶ Universal Mobile Telecommunication system
- It is used to send broadcast intents when the network connectivity manager changes
- It will attempt to switch over to other networks when the connectivity is lost. It provides an API to query the, state of the available networks.

Methods in **ConnectivityManager** class

1. **public NetworkInfo getActiveNetworkInfo():** it returns an object of type network info, which is currently active device.
2. **public NetworkInfo[ ] getAllNetwrokinfo();** it returns the network connections registered with mobile.
3. **public NetworkInfo gerNetworkInfo(int** networkType**);** it returns an object of type network info, which describe the network connection of the passed network type.
4. **public int getNetworkPreferance();** it returns the type of network to be used when multiple network connections are registered with device.
5. **public void setNetworkPreference(int networkPreference);** it fixes a reference level on the network connection
6. **public boolean isNetworkTypeValid(int** networkType**);** it retuns a Boolean value, true the network is valid.

Constants defined in the connectivity manager class

TYPE_MOBILE
It is the default mobile data connection, when it is active all the data traffic will use this connection by default.

TYPE_MOBILE_DUN:
It is DUN specific mobile data connection. This is used by the applications performing a dial up networking connection

TYPE_MOBILE_HIPRI:
It is a high priority mobile data connection

TYPE_MOBILE_MMS:

Multimedia service(MMS)
It is a specific mobile
This is used by the applications which want to talk to the MMS servers.

TYPE_WIFI
It is used to the default wifi connection. When it is active all the data traffic will be suing this connection by default.

TYPE_WUIMAX: (worldwide intrapability for microwave access)
It is used to network protocol.
It is by default wuimax data connection.

WifiManager: This class provides the basic API for managing wifi connectivity. An instance of this class can be created by calling.

Context.getSystemService (Context.WIFI_SERVICE);
WifiManager wm = (WifiManager)gerSystemService
(Context. WIFI_SERVICE)

The basic functionality of this class ;

1. It lists out all the configured networks on the mobile. The list items can be view updated.
2. It shows the currently active wifi network. The dynamic information about the currently active network can be queried.
3. It's scan the access points available and it contains information about access point to connection
4. It defines various intend actions, that are to be broad casted upon changes wifi state.

CONSTANTS DEFINED IN THE WIFI MANAGER CLASS

1. **WIFI_STATE_DESABLE**
2. **WIFI_STATE_DESABLING**
3. **WIFI_STATE_UNKNOWN**: wifi is an unknown state
4. **WIFI_STATE_ENABLED**: wifi is enabled.
5. **WIFI_STATE_ENABLING**: wifi is currently being enabling.
6. **EXTRA_BASSID** (Basic service set Identifier)
   It is the key giving BSSID of the access point to which we are connected.
7. **EXTRA_PREVIOUS_WIFI_STATE**: it is the previous wifi state.

8. **EXTRA_WIFI_STATE** It defines whether wifi is enable, classified, enabling, disabling are unknown.

Methods in the WifiManager class

1. **public boolean startScan():** It requests scan for access points. APN (Access point names).
2. **public boolean setWifiEnable(true)**
3. **public int getWifiState():**
4. **public boolean isWifiEnabled():** it returns whether wifi is enabled/disabled.
5. **public List<ScanResults> getScanResults():** it returns the results of the latest access point scan
6. **public WifiInfo getConnectionInfo():** it returns the dynamic information about the current wifi connection
7. **public List<WifiConfiguration> getConfiguredNetworks():** it returns a list of all the network confirgured
8. **public boolean disableNetwork():** it disables the configure network connection.
9. **public boolean disconnect():** it disconnects from the currently active access point.
10. **public boolean reassociate():** it reconnect currently access point even if we are already connected.
11. **public boolean reconnection:** it reconnects to the currently active access point if we are already disconnected.

Android Wifi, How to scan wireless networks?

Scan for wireless networks in the current area:

```
package com.android.wifi;

import java.util.List;
import android.app.Activity;
import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;
import android.content.IntentFilter;
import android.net.wifi.ScanResult;
import android.net.wifi.WifiManager;
import android.os.Bundle;
```

```
import android.view.Menu;
import android.view.MenuItem;
import android.widget.TextView;

public class WifiTester extends Activity {
 TextView mainText;
 WifiManager mainWifi;
 WifiReceiver receiverWifi;
 List<ScanResult> wifiList;
 StringBuilder sb = new StringBuilder();

 public void onCreate(Bundle savedInstanceState) {
  super.onCreate(savedInstanceState);
  setContentView(R.layout.main);
  mainText = (TextView) findViewById(R.id.mainText);
  mainWifi = (WifiManager) getSystemService(Context.WIFI_SERVICE);
  receiverWifi = new WifiReceiver();
  registerReceiver(receiverWifi, new IntentFilter(
    WifiManager.SCAN_RESULTS_AVAILABLE_ACTION));
  mainWifi.startScan();
  mainText.setText("\nStarting Scan...\n");
 }

 public boolean onCreateOptionsMenu(Menu menu) {
  menu.add(0, 0, 0, "Refresh");
  return super.onCreateOptionsMenu(menu);
 }

 public boolean onMenuItemSelected(int featureId, MenuItem item) {
  mainWifi.startScan();
  mainText.setText("Starting Scan");
  return super.onMenuItemSelected(featureId, item);
 }

 protected void onPause() {

  unregisterReceiver(receiverWifi);
  super.onPause();
 }

 protected void onResume() {

  registerReceiver(receiverWifi, new IntentFilter(
    WifiManager.SCAN_RESULTS_AVAILABLE_ACTION));
  super.onResume();
 }
```

```
class WifiReceiver extends BroadcastReceiver {

  public void onReceive(Context c, Intent intent) {
  sb = new StringBuilder();
  wifiList = mainWifi.getScanResults();
  for(int i = 0; i < wifiList.size(); i++){

   sb.append(new Integer(i+1).toString() + ".");
   sb.append((wifiList.get(i)).toString());
   sb.append("\n");
  }
  mainText.setText(sb);
  }
 }
}
```

BlueTooth.

- Bluetooth is a communication protocol, which is based on Radio Technology & which is designed for short range, low bandwidth communications / connections(Bandwidth--. Channels(Ex:256 KB/1 Mb));
- Connections are two types.
    1) Ad-hoc Connection
    2) Encripted Connection

Piconet:
A master device with other devices (friendly namely)
We can connect with maximum six slave devices by using Master devices (In Piconet 6 maximum devices)We can count with number of devices by using slave devices (It acts as a master)

Bluetooth:

- Settings
- Discovering the devices
- Connecting to devices/Pairing with device
- Transferring the data
- Connections are 2 types (Normal connection or Adhoc Connections) and Enterpitid connection(By default Android supports).

The Major Class:

1. Accessing the Bluetooth h/w of the parent device.

BluetoothAdapter class (To access the local Bluetooth hardware after that remote Bluetooth will be accessed)

2. Discovering the remote device

   BluetoothDevice class

3. Sending/Transfering the data in one direction acting as a client

   BluetoothSocket class

4. Transfering the data in bidirectional.

5. BluetoothServerSocket class

   2 Permissions are required:
   android.permission.BLUETOOTH(Accessing the Bluetooth permission).
   android.permission.BLUETOOTH_ADMIN(Clearing the Bluetooth permission).
   In the element<user-permission> tag.

Ad-hoc Connection: Using this connection communication can make b/w un paired devices.
Encripted Connection: Connections can be made b/w paired devices.

- The classes which are used to handle the Bluetooth devices and connections are
- BluetoothAdapter: The local BT device is controlled using this class. To access the DefaultBluetoothAdapter, getDefaultAdapter() method can be called on the local devic e.
- To read the BluetoothAdapter properties, to start discovery/to find the paired device, one should specify the BLUETOOTH Manifest permission <users-permission>.
- To modify the any of the local Device properties, BLUETOOTH_ADMIN manifest permission required.
- To access the BluetoothAdapter properties the Bluetooth should be switched on or enabled. We can check the state of the Bluetooth h/w using the isEnabled() method.
- The friendly name of the BluetoothAdapter can be returned using the getName() method.
- The Address of the BluetoothAdapter can be retrived using the getAddress() method. By default the BluetoothAdapter is TURNED_OFF.
- The state of the BluetoothAdapter can be retrived by using the getState() method. It will be return one of the following constants defined in the BluetoothAdapter class.

  STATE_ON
  STATE_OFF
  STATE_TURNING_ON
  STATE_TURNING_OFF

- The process of 2 devices finding each other in order to connect is called Discovering.
  1. Step1: - For 2 devices to pair with each other they need to be discovered by each other.
  2. Step2: - After discovering each other the BluetoothAdapter of Both the devices is paired with each other.

3. Step3: - After paring both the devices should establish a socket connection for communication.

- The BluetoothAdapters discoverability is indicating by scan mode. We can retrieve the scan mode of Adapter on the BluetoothAdapter object by calling getScanMode() method. This method will return the constants defined in the BluetoothAdapter class.
- There are 2 types of Scan modes:
  1. Inquiry Scan.
  2. Page Scan.
- Inquiry Scan: If this mode is enable the device is discourable from any Bluetooth device. This is performing a discovery Scan.
- Page Scan: If this mode is enabled devices can be discovered that are previously connected and paired with the local device. In this mode new devices cannot be discovered.

Constants:

- ✓ SCAN_MODE_CONNECTABLE_DISCOVERABLE : If this constant is return, It means that both the scan modes are enabled.
- ✓ SCAN_MODE_CONNECTABLE: If this is enabled, it means only page scan mode is enabled.
- ✓ SCAN_MODE_NONE: It means discoverability is turned off. It means no other devices, which are process of discovery scan find the local device.
- ✓ By default Android devices will have disoverability is disabled.
- ✓ If you want to make it discoverable explicit permission should be asked from the user. By default discoverability will be enabled for 2 minutes.

Discovering Remote Devices: To discover remote dev ice the local BluetoothAdapter is to be checked, whether it is already performing a discovering scan. It is checked using the isDiscovering() method.

- To start the discovery process call the method startDiscovery () method on the BluetoothAdapter class.
- To cancel a Discovery call the cancelDiscovery () method on the BluetoothAdaptor class. The discovery process is **asynchronous**.
- BluetoothSocket → Client mode connection.
- BluetoothServerSocket → Server mode connection.

1. Swithc on Bluetooth.
2. Searching particular device to send or receive data.
3. Send a request.
4. Pairing the device.

- Here we are sending the data(Transfers song), then how much % sending, this we user ProgressDialog and we can communicate by using BroadcastReciever. After that we registerReciever() method we can override registerReciever(BroadcastReciever object, Intent obj);
- To start a discovery then we use:ACTION_DISCOVERY_STARTED
- To finish a discovery then we use To start a discovery then we use:ACTION_DISCOVERY_FINISH

//Initiating a service
BluetoothSocket bs = createRfcommSocketToServiceRecord ();[Radio frequency communication]
listenUsingRfcommwithServiceRecord(string name, string UUID); [Universally Uniquie Identification]
UUID uid= UUID.parse("Hardware Address");

- ✓ If you are willing to pair a device to that connection yes/no. yes means accept()method will be called.
- ✓ getRemoteDevice(); → It is for which devices are available in the device.
- ✓ getBandedDevices(); → It returns set, It shows all attached devices it shows.

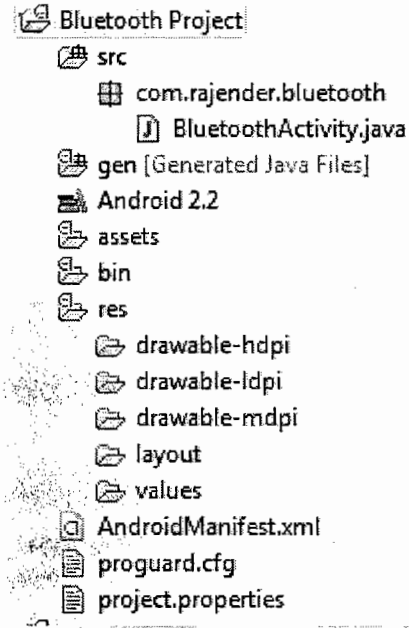
Bluetooth Communicaton

Bluetooth communications work among Rfcom, the Bluetooth RadioFrequency communication protocol. To establish a Rfcom connection, the following classes are required.

1. BluetoothSocket: It is used in creating a new client socket to connect to a listening BluetoothServerSocket. It is returned by the ServerSocket once a connection is established. Once the connection is made, Bluetooth Sockets are used on both end to Transfer data streams.
2. BluetoothServerSocket: It is used to establish a listening socket for initially the link b/w 2 devices. To establish a connection one device acts as a server to listen for and accept in coming connection request.
    - Opening a BluetoothServerSocket listener: A BluetoothServerSocket is used to listen for BluetoothSocket connection request from remote Bluetooth devices.
    - To listen for incoming connection request call the ListenUsing RfcomwithServiceRecord() method on the BluetoothAdapter class. It takes 2 arguments
        1. It is the name to identify the server and this will return a BluetoothServerSocket object.
        2. Universally Uniquie Identification(UUID).

- To start listening for connections, call accept() on the ServerSocket optionally passing in a timeout duration.
- getRemoteDevice() method on your on the BluetoothAdapter objet returns the RemoteDevice. If you want to connect to.
- To find the set of currently paired devices to call getBoundedDevice() method on local BluetoothAdapter.
- To start the discovery process ACTIO_DISCOVERY_STARTED Intent action is passed, to finish the discovery process want to identify ACTIO_DISCOVERY_FINISHED called.

Bluetooth Project



BluetoothActivity.java

```
package com.rajender.bluetooth;

import java.util.Set;
import android.app.Activity;
import android.bluetooth.BluetoothAdapter;
import android.bluetooth.BluetoothDevice;
import android.os.Bundle;
import android.widget.Toast;

public class BluetoothActivity extends Activity {
   BluetoothAdapter ba;
        @Override
   public void onCreate(Bundle b) {
      super.onCreate(b);
      setContentView(R.layout.main);
      ba=BluetoothAdapter.getDefaultAdapter();
```

```
ba.setName("My Bluetooth");
String name=ba.getName();

Toast.makeText(getApplicationContext(),
        "Bluetooth Name is : "+name, Toast.LENGTH_LONG).show();

String address=ba.getAddress();
Toast.makeText(getApplicationContext(),
        "Bluetooth Address is : "+address, Toast.LENGTH_LONG).show();

int state=ba.getState();

switch (state) {
case BluetoothAdapter.STATE_ON:
        Toast.makeText(getApplicationContext(),
        "Bluetooth State is : STATE_ON", Toast.LENGTH_LONG).show();
        break;

case BluetoothAdapter.STATE_OFF:
        Toast.makeText(getApplicationContext(),
        "Bluetooth State is : STATE_OFF",
   Toast.LENGTH_LONG).show();
        break;
case BluetoothAdapter.STATE_TURNING_OFF:
        Toast.makeText(getApplicationContext(),
        "Bluetooth State is : STATE_TURNING_OFF",
   Toast.LENGTH_LONG).show();
        break;
case BluetoothAdapter.STATE_TURNING_ON:
        Toast.makeText(getApplicationContext(),
        "Bluetooth State is : STATE_TURNING_ON",
   Toast.LENGTH_LONG).show();
        break;
   }

int scanMode=ba.getScanMode();

switch (scanMode) {
case BluetoothAdapter.SCAN_MODE_CONNECTABLE:
        Toast.makeText(getApplicationContext(),
        "Bluetooth Scan Mode is : SCAN_MODE_CONNECTABLE",
        Toast.LENGTH_LONG).show();
        break;

case BluetoothAdapter.SCAN_MODE_CONNECTABLE_DISCOVERABLE:
```

```
            Toast.makeText(getApplicationContext(),
            "Bluetooth Scan Mode is :
        SCAN_MODE_CONNECTABLE_DISCOVERABLE",
            Toast.LENGTH_LONG).show();
            break;

    case BluetoothAdapter.SCAN_MODE_NONE:
            Toast.makeText(getApplicationContext(),
            "Bluetooth Scan Mode is : SCAN_MODE_NONE",
            Toast.LENGTH_LONG).show();
            break;
      }

    boolean discovery=ba.startDiscovery();
    Toast.makeText(getApplicationContext(),"Bluetooth Discovery is :
      "+discovery,   Toast.LENGTH_LONG).show();

       // Listing paired devices
        Set<BluetoothDevice> devices =ba.getBondedDevices();
        for (BluetoothDevice device : devices) {
            Toast.makeText(getApplicationContext(),
                "BondedDevices is : "+device,
                Toast.LENGTH_LONG).show();

        }
    }
}
```

AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
  package="com.rajender.bluetooth"
  android:versionCode="1"
  android:versionName="1.0" >

  <uses-sdk android:minSdkVersion="8" />

      <uses-permission android:name  = "android.permission.BLUETOOTH"/>
      <uses-permission android:name  = "android.permission.BLUETOOTH_ADMIN"/>

  <application
    android:icon="@drawable/ic_launcher"
    android:label="@string/app_name" >
    <activity
      android:name="com.rajender.bluetooth.BluetoothActivity"
      android:label="@string/app_name" >
      <intent-filter>
```

```
            <action android:name="android.intent.action.MAIN" />

            <category android:name="android.intent.category.LAUNCHER" />
        </intent-filter>
    </activity>
</application>

</manifest>
```

Media Components

- **Camera**

The Camera class is used to set image capture settings, start/stop preview, snap pictures, and retrieve frames for encoding for video. This class is a client for the Camera service, which manages the actual camera hardware.

To access the device camera, you must declare the CAMERA permission in your Android Manifest

<uses-permission android:name="android.permission.CAMERA"/>

To take pictures with this class, use the following steps:

1. Obtain an instance of Camera from open(int).
2. Get existing (default) settings with getParameters().

3. If necessary, modify the returned Camera.Parameters object and call setParameters(Camera.Parameters).
4. If desired, call setDisplayOrientation(int).
5. **Important**: Pass a fully initialized SurfaceHolder to setPreviewDisplay(SurfaceHolder). Without a surface, the camera will be unable to start the preview.
6. **Important**: Call startPreview() to start updating the preview surface. Preview must be started before you can take a picture.
7. When you want, call takePicture(Camera.ShutterCallback, Camera.PictureCallback, Camera.PictureCallback, Camera.PictureCallback) to capture a photo. Wait for the callbacks to provide the actual image data.
8. After taking a picture, preview display will have stopped. To take more photos, call startPreview() again first.
9. Call stopPreview() to stop updating the preview surface.
10. **Important:** Call release() to release the camera for use by other applications. Applications should release the camera immediately in onPause() (and re-open() it in onResume()).

This grants access to the Camera Service. The Camera class lets you adjust camera settings, take pictures, and manipulate streaming camera previews.
To access the Camera Service, use the static open method on the Camera class. When your application has finished with the camera, remember to relinquish your hold on the Service by calling release following the simple use pattern shown in the code snippet below:

```
Camera camera = Camera.open();
[ ... Do things with the camera ... ]
camera.release();
```

Controlling Camera Settings

The current camera settings are available as a Camera.Parameters object. Call the getParameters method on the Camera to access the current parameters.
You can use the set* methods on the returned Parameters to modify the settings. To apply changes, call setParameters, passing in the modified values as shown below:

```
Camera.Parameters parameters = camera.getParameters();
parameters.setPictureFormat(PixelFormat.JPEG);
camera.setParameters(parameters);
```

The Camera Parameters can be used to specify the image and preview size, image format, and preview frame rate.

Using the Camera Preview

Access to the camera's streaming video means that you can incorporate live video into your applications. Some of the most exciting early Android applications have used this functionality as the basis for augmenting reality.

The camera preview can be displayed in real time onto a Surface, as shown in the code snippet below:

```
camera.setPreviewDisplay(mySurface);
camera.startPreview();
[ ... ]
camera.stopPreview();
```

You can also assign a PreviewCallback to be fired for each preview frame, allowing you to manipulate or display each preview frame individually. Call the setPreviewCallback method on the Camera object, passing in a new PreviewCallback implementation overriding the onPreviewFrame method as shown here:

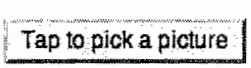
```
camera.setPreviewCallback(new PreviewCallback() {
  public void onPreviewFrame(byte[] _data, Camera _camera) {
  // TODO Do something with the preview image.
  }
});
```

Taking a Picture

Take a picture by calling takePicture on a Camera object, passing in a ShutterCallback and PictureCallback implementations for the RAW and JPEG-encoded images. Each picture callback will receive a byte array representing the image in the appropriate format, while the shutter callback is triggered immediately after the shutter is closed.

```
private void takePicture() {
 camera.takePicture(shutterCallback, rawCallback, jpegCallback);
 }
 ShutterCallback shutterCallback = new ShutterCallback() {
 public void onShutter() {
// TODO Do something when the shutter closes.
 }
};
 PictureCallback rawCallback = new PictureCallback() {
  public void onPictureTaken(byte[] _data, Camera _camera) {
  // TODO Do something with the image RAW data.
  }
};
 PictureCallback jpegCallback = new PictureCallback() {
  public void onPictureTaken(byte[] _data, Camera _camera) {
   // TODO Do something with the image JPEG data.
  }
};
```

ImagePicker



Use an image picker component to choose an image from your image gallery.

An image picker is a kind of button. When the user taps an image picker, the device's image gallery appears, and the user can choose an image. After the user picks an image, the property ImagePath is set to a text string that represents that image. You can then use that result, for example, to set the image of a button.

Properties

ImagePath
: The image the user chose, represented as a text string that gives the location of the images.

Enabled
: If true, image picker can be used.

Alignment
: Left, center, or right.

BackgroundColor
: Color for image picker background.

Enabled
: If set, user can tap image picker to cause action.

FontBold
: If set, image picker button text is displayed in bold.

FontItalic
: If set, image picker button text is displayed in italics.

FontSize
: Point size for image picker button text.

FontTypeface
: Font family for image picker button text.

Height
: image picker button height (y-size).

Width
: image picker button width (x-size).

Image
: Image to display on image picker button.

Text
: Text to display on image picker button.

TextColor
: Color for image picker button text.

Events

afterPicking()
: User selected an item from the image picker.

beforePicking()
: User has tapped the image picker but hasn't yet selected an item.

gotFocus()
: Image picker became the focused component.

lostFocus()

Image picker is no longer the focused component.

Player

Use a player component to play an audio or video file, or to vibrate the phone.

Player is a non-visible component that plays audio or video and controls phone vibration. The name of a media file is specified in the Source property, which can be set in the Designer or in the Blocks Editor. The length of time for a vibration is specified in the Blocks Editor in milliseconds (thousandths of a second).

Use a player component for playing long sound files, video files, and vibrating the phone. For playing short sound files, such as sound effects, use a Sound component instead.

Properties

Source
Audio or video file associated with this player.

Methods

pause()
Pauses playing the audio or video file.
start()
Starts playing the audio or video file.
stop()
Stops playing the audio or video file.
vibrate(number milliseconds)
Activate the phone's vibration motor for the given number of milliseconds.

Sound

Use a sound component to play an audio file, or to vibrate the phone.

Sound is a non-visible component that plays sound files and vibrates for the number of milliseconds (thousandths of a second) specified in the Blocks Editor. The name of the sound file to play can be specified either in the Designer or in the Blocks Editor.

This component is best for short sound files, such as sound effects, while the Player component is more efficient for longer sounds, such as songs.

Properties

Source
: Audio file associated with this sound.

MinimumInterval
: Minimum time before sound is repeated.

Methods

pause()
: Pauses playing the audio file.

play()
: Starts playing the audio file.

resume()
: Resumes playing a paused audio file.

stop()
: Stops playing the audio file.

vibrate(number millisecs)
: Activate the phone's vibration motor for the given number of milliseconds.

VideoPlayer

Use a VideoPlayer component to play a video file.

Video player is a media component that plays videos. A video player appears in your app as a rectangle. If the user taps the rectangle, media controls appear: play/pause, skip ahead, and skip backward. Your app can control playback behavior by calling the Start, Pause, and Stop methods.

Video files should be in Windows Media Video (.wmv), 3GPP (.3gp), or MPEG-4 (.mp4) format. For more details about formats, see Android Supported Media Formats.

App Inventor accepts video files up to 1 MB in size and limits the total size of an app to 5 MB, not all of which is available for media files. If your media files are too large, you might get errors when packaging or installing your app, in which case you should reduce the number of media files or their sizes. Video editing software, such as Windows Movie Maker or Apple iMovie, can help you decrease the size of videos by shortening them or re-encoding them into more compact formats.

Properties

Source
: Video file associated with this player.

Visible
: If set, VideoPlayer is visible.

Methods

pause()

Pauses playing the video file.

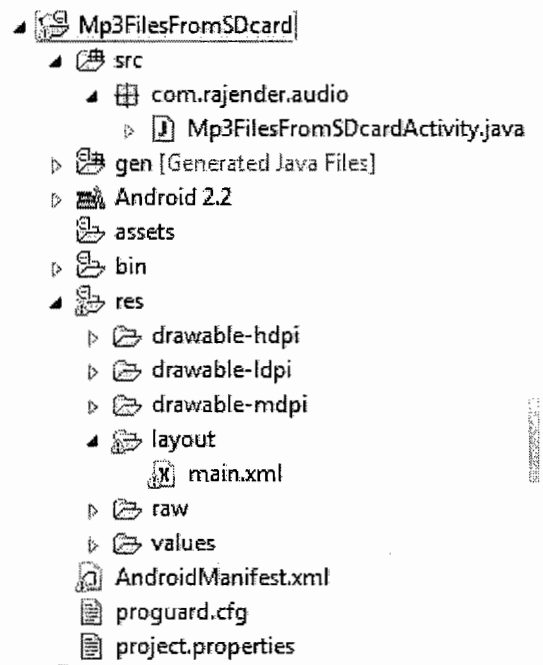
start()

Starts playing the video file.

stop()

Stops playing the video file.

Audio Program



main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
   android:orientation="vertical"
   android:layout_width="fill_parent"
   android:layout_height="fill_parent"
   android:background="#00ffff">

   <Spinner
      android:id="@+id/spinner"
      android:layout_width="match_parent"
      android:layout_height="wrap_content" />

   <Button
      android:onClick="audioFunction"
      android:id="@+id/playButton"
      android:layout_width="fill_parent"
```

```
        android:layout_height="wrap_content"
        android:text="Play" />

    <Button
        android:onClick="audioFunction"
        android:id="@+id/pauseButton"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="Pause" />
    <Button
        android:onClick="audioFunction"
        android:id="@+id/stopButton"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="Stop" />
</LinearLayout>
```

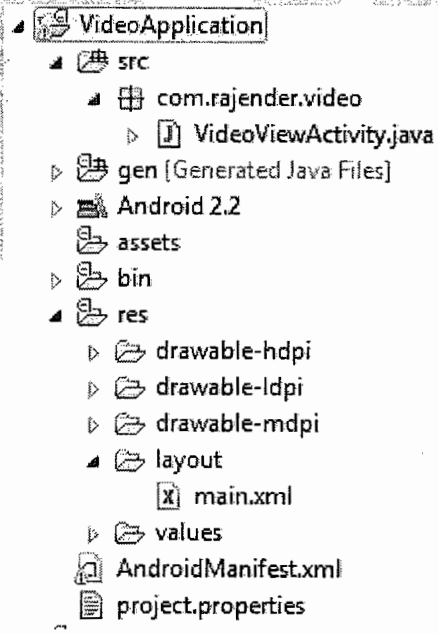
Mp3FilesFromSDcardActivity.java

```
package in.com.sdcard;

import java.io.File;
import java.util.ArrayList;
import android.app.Activity;
import android.media.MediaPlayer;
import android.os.Bundle;
import android.os.Environment;
import android.view.View;
import android.widget.AdapterView;
import android.widget.AdapterView.OnItemSelectedListener;
import android.widget.ArrayAdapter;
import android.widget.Spinner;

public class Mp3FilesFromSDcardActivity extends Activity
{

 ArrayList<String> al;
        Spinner sp ;
        String path,selectedSong;
        MediaPlayer player;

    @Override
    public void onCreate(Bundle bundle)
    {
        super.onCreate(bundle);
        setContentView(R.layout.main);
```

```
al = new ArrayList<String>();
sp = (Spinner)findViewById(R.id.spinner);
player = new  MediaPlayer();
player.setLooping(true);

        path = Environment.
                        getExternalStorageDirectory().
                                                getAbsolutePath();
        File file = new File(path);
     File[] listOfFiles = file.listFiles();

    for (int i = 0; i < listOfFiles.length; i++)
    {
        if (listOfFiles[i].isFile())
        {
                if (listOfFiles[i].getName().endsWith((".mp3")))
                {
                        al.add(listOfFiles[i].getName());
                }
        }
}

   ArrayAdapter<String> adapter = new ArrayAdapter<String>
                (this,android.R.layout.simple_dropdown_item_1line, al);
 sp.setAdapter(adapter);

 sp.setOnItemSelectedListener(new OnItemSelectedListener()
 {
                public void onItemSelected(AdapterView<?> av,
                                View v,int possition, long itemId)
                {
                selectedSong=av.getItemAtPosition(possition).toString();
                }

                public void onNothingSelected(AdapterView<?> av) {
                }
 });
}


public void audioFunction(View v) {

    try
                {
                        player.setDataSource(path+"/"+selectedSong);
                        player.prepare();
```

```
                }
                catch (Exception e)
            {
                e.printStackTrace();
            }
        switch (v.getId()) {
            case R.id.playButton:
                player.start();
                break;

            case R.id.pauseButton:
                player.pause();
                break;
            case R.id.stopButton:
                player.stop();
                player.reset();
                break;
            }

    }
}
```

Video Project



VideoViewActivity.java

```
package com.rajender.video;

import android.app.Activity;
import android.os.Bundle;
import android.os.Environment;
import android.widget.MediaController;
import android.widget.VideoView;
import java.io.File;

public class VideoViewActivity extends Activity {
        VideoView video;
        MediaController       ctlr;
        @Override
        public void onCreate(Bundle b) {
                super.onCreate(b);
                setContentView(R.layout.main);
                String path=
                Environment.getExternalStorageDirectory()
                                                        .getAbsolutePath();
                //File path=new File(path,"myvideo.mp4");
                File file=new File(path+"/myvideo.mp4");
                if (file.exists()) {
                 video=(VideoView)findViewById(R.id.video);

                 video.setVideoPath(file.getAbsolutePath());
                 //video.setVideoURI(Uri uri);//From Internet
                        ctlr=new MediaController(this);
                        ctlr.setMediaPlayer(video);

                        video.setMediaController(ctlr);
                        video.requestFocus();
                        video.start();
                }
        }
}
```

main.xml

```
<?xml version="1.0" encoding="utf-8"?>
        <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
        android:orientation="vertical"
        android:layout_width="fill_parent"
        android:layout_height="fill_parent" >

        <VideoView
                android:id="@+id/video"
                android:layout_width="fill_parent"
                android:layout_height="fill_parent"/>

</LinearLayout>
```

Data Storage

The ability to store data locally on the mobile device is a critical function for mobile applications that are required to maintain essential information across application-executions or the lifetime of the application. As a developer, you constantly need to store information such as user preferences or application configurations. You must also decide if you need to tap internal or external storage, depending on characteristics, such as access visibility, or if you need to handle more complex, structured types of data. Follow along in this article to learn about Android data storage APIs, specifically the preferences, SQLite, and the internal and external memory APIs.

Preferences :- Preferences is a lightweight mechanism to store and retrieve key-value pairs of primitive data types. It is private data and permanent storage.

Files :-You can store your data in files on your mobile phone, or in a removable storage medium. It is also private data and temporary storage.

SD Card :- It is external storage stored on SD Card. It is public data and permanent storage.

Databases :- Android Api supports SQLite databases. All databases, SQLite and others, are stored on the device in /data/data/package_name/databases.

Using Shared Preferences

The SharedPreferences class provides a general framework that allows you to save and retrieve persistent key-value pairs of primitive data types. You can use SharedPreferences to save any primitive data: booleans, floats, ints, longs, and strings. This data will persist across user sessions (even if your application is killed).

User Preferences

Shared preferences are not strictly for saving "user preferences," such as what ringtone a user has chosen. If you're interested in creating user preferences for your application.

To get a SharedPreferences object for your application, use one of two methods:

- getSharedPreferences() - Use this if you need multiple preferences files identified by name, which you specify with the first parameter.
- getPreferences() - Use this if you need only one preferences file for your Activity. Because this will be the only preferences file for your Activity, you don't supply a name.

To write values:

1. Call edit() to get a SharedPreferences.Editor.
2. Add values with methods such as putBoolean() and putString().
3. Commit the new values with commit()

SharedPreferencesExample

SharedPreferenceActivity.java

```
package com.preferences;
import android.app.Activity;
import android.content.SharedPreferences;
import android.os.Bundle;
import android.widget.TextView;

public class SharedPreferenceActivity extends Activity {
   @Override
   public void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.main);
      SharedPreferences pref=getPreferences(MODE_PRIVATE);
      int counter=pref.getInt("count",0);
      TextView tv=(TextView)findViewById(R.id.textView);
      tv.setText("This application has been executed "+counter+" times.");
      SharedPreferences.Editor editor=pref.edit();
       editor.putInt("count", ++counter);
      editor.commit();
   }
}
```

main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
   android:orientation="vertical"
   android:layout_width="fill_parent"
   android:layout_height="fill_parent"
   >
<TextView
   android:layout_width="fill_parent"
   android:layout_height="wrap_content"
   android:id="@+id/textView"
   />
```

</LinearLayout>

Files(Using the Internal Storage)

You can save files directly on the device's internal storage. By default, files saved to the internal storage are private to your application and other applications cannot access them (nor can the user). When the user uninstalls your application, these files are removed.

To create and write a private file to the internal storage:

1. Call openFileOutput() with the name of the file and the operating mode. This returns a FileOutputStream.
2. Write to the file with write().
3. Close the stream with close().

For example:

```
String FILENAME = "hello_file";
String string = "hello world!";

FileOutputStream fos = openFileOutput(FILENAME, Context.MODE_PRIVATE);
fos.write(string.getBytes());
fos.close();
```

MODE_PRIVATE will create the file (or replace a file of the same name) and make it private to your application. Other modes available are: MODE_APPEND, MODE_WORLD_READABLE, and MODE_WORLD_WRITEABLE.

To read a file from internal storage:

1. Call openFileInput() and pass it the name of the file to read. This returns a FileInputStream.
2. Read bytes from the file with read().
3. Then close the stream with close().

Tip: If you want to save a static file in your application at compile time, save the file in your project `res/raw/` directory. You can open it with openRawResource(), passing the `R.raw.<filename>` resource ID. This method returns an InputStream that you can use to read the file (but you cannot write to the original file).

Saving cache files If you'd like to cache some data, rather than store it persistently, you should use getCacheDir() to open a File that represents the internal directory where your application should save temporary cache files.

When the device is low on internal storage space, Android may delete these cache files to recover space. However, you should not rely on the system to clean up these files for you. You

should always maintain the cache files yourself and stay within a reasonable limit of space consumed, such as 1MB. When the user uninstalls your application, these files are removed.

Other useful methods

getFilesDir()

Gets the absolute path to the filesystem directory where your internal files are saved.

getDir()

Creates (or opens an existing) directory within your internal storage space.

deleteFile()

Deletes a file saved on the internal storage.

fileList()

Returns an array of files currently saved by your application.

FileActivity.java

```
package com.files;

import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.InputStreamReader;
import java.io.OutputStreamWriter;

import android.app.Activity;
import android.os.Bundle;
import android.widget.TextView;

public class FileActivity extends Activity {
	@Override
	public void onCreate(Bundle savedInstanceState) {
		super.onCreate(savedInstanceState);
		setContentView(R.layout.main);

		try {
			BufferedWriter writer = new BufferedWriter(new OutputStreamWriter(
					openFileOutput("myfile",
MODE_WORLD_WRITEABLE)));
			writer.write("Welcome to Android1" +"\n");
			writer.write("Welcome to Android2" +"\n");
```

```
            writer.write("Welcome to Android3" +"\n");
            writer.write("Welcome to Android4" +"\n");
            writer.write("Welcome to Android5" +"\n");
            writer.close();
        } catch (Exception e) {
            e.printStackTrace();
        }
        try {
            BufferedReader input = new BufferedReader(new InputStreamReader(
                        openFileInput("myfile")));
            String line;
            StringBuffer sb = new StringBuffer();
            while ((line = input.readLine()) != null) {
                sb.append(line + "\n");
            }
            TextView textView = (TextView) findViewById(R.id.result);

            textView.setText(sb.toString());

        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

SD Card(Using the External Storage)

Every Android-compatible device supports a shared "external storage" that you can use to save files. This can be a removable storage media (such as an SD card) or an internal (non-removable) storage. Files saved to the external storage are world-readable and can be modified by the user when they enable USB mass storage to transfer files on a computer.

Caution: External files can disappear if the user mounts the external storage on a computer or removes the media, and there's no security enforced upon files you save to the external storage. All applications can read and write files placed on the external storage and the user can remove them.

Checking media availability

Before you do any work with the external storage, you should always call getExternalStorageState() to check whether the media is available. The media might be mounted to a computer, missing, read-only, or in some other state. For example, here's how you can check the availability:

```
boolean mExternalStorageAvailable = false;
boolean mExternalStorageWriteable = false;
String state = Environment.getExternalStorageState();

if (Environment.MEDIA_MOUNTED.equals(state)) {
    // We can read and write the media
    mExternalStorageAvailable = mExternalStorageWriteable = true;
} else if (Environment.MEDIA_MOUNTED_READ_ONLY.equals(state)) {
    // We can only read the media
    mExternalStorageAvailable = true;
    mExternalStorageWriteable = false;
} else {
    // Something else is wrong. It may be one of many other states, but all we need
    //  to know is we can neither read nor write
    mExternalStorageAvailable = mExternalStorageWriteable = false;
}
```

This example checks whether the external storage is available to read and write. The getExternalStorageState() method returns other states that you might want to check, such as whether the media is being shared (connected to a computer), is missing entirely, has been removed badly, etc. You can use these to notify the user with more information when your application needs to access the media.

Accessing files on external storage

If you're using API Level 8 or greater, use getExternalFilesDir() to open a File that represents the external storage directory where you should save your files. This method takes a `type` parameter that specifies the type of subdirectory you want, such as DIRECTORY_MUSIC and DIRECTORY_RINGTONES (pass `null` to receive the root of your application's file directory). This method will create the appropriate directory if necessary. By specifying the type of directory, you ensure that the Android's media scanner will properly categorize your files in the system (for example, ringtones are identified as ringtones and not music). If the user uninstalls your application, this directory and all its contents will be deleted.

If you're using API Level 7 or lower, use getExternalStorageDirectory(), to open a File representing the root of the external storage. You should then write your data in the following directory:

/Android/data/<*package_name*>/files/

The <*package_name*> is your Java-style package name, such as "`com.example.android.app`". If the user's device is running API Level 8 or greater and they uninstall your application, this directory and all its contents will be deleted.

Saving files that should be shared

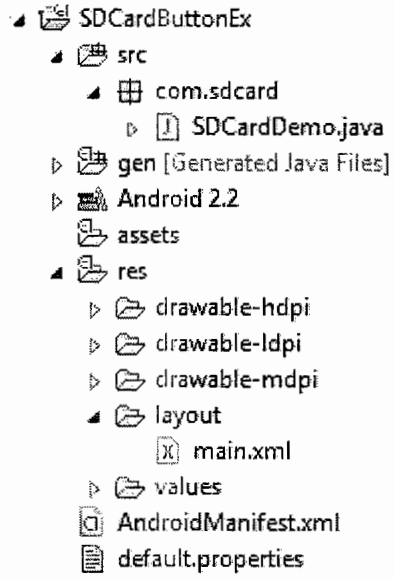
If you want to save files that are not specific to your application and that should *not* be deleted when your application is uninstalled, save them to one of the public directories on the external storage. These directories lay at the root of the external storage, such as `Music/`, `Pictures/`, `Ringtones/`, and others.

In API Level 8 or greater, use getExternalStoragePublicDirectory(), passing it the type of public directory you want, such as DIRECTORY_MUSIC, DIRECTORY_PICTURES, DIRECTORY_RINGTONES, or others. This method will create the appropriate directory if necessary.

If you're using API Level 7 or lower, use getExternalStorageDirectory() to open a File that represents the root of the external storage, then save your shared files in one of the following directories:

- `Music/` - Media scanner classifies all media found here as user music.
- `Podcasts/` - Media scanner classifies all media found here as a podcast.
- `Ringtones/` - Media scanner classifies all media found here as a ringtone.
- `Alarms/` - Media scanner classifies all media found here as an alarm sound.
- `Notifications/` - Media scanner classifies all media found here as a notification sound.
- `Pictures/` - All photos (excluding those taken with the camera).
- `Movies/` - All movies (excluding those taken with the camcorder).
- `Download/` - Miscellaneous downloads.

SDCard Example :-



SDCardDemo.java

```
package com.sdcard;

import java.io.*;
import android.app.Activity;
import android.os.Bundle;
import android.view.*;
import android.view.View.OnClickListener;
import android.widget.*;

public class SDCardDemo extends Activity {
        // GUI controls
        EditText txtData;
        Button btnWriteSDFile;
        Button btnReadSDFile;
        Button btnClearScreen;
        Button btnClose;

        @Override
        public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
```

```
setContentView(R.layout.main);
// bind GUI elements with local controls
txtData = (EditText) findViewById(R.id.txtData);
txtData.setHint("Enter some lines of data here...");

btnWriteSDFile = (Button) findViewById(R.id.btnWriteSDFile);
btnWriteSDFile.setOnClickListener(new OnClickListener() {

public void onClick(View v) {
        // write on SD card file data in the text box
        try {
                File myFile = new File("/sdcard/mysdfile.txt");
                myFile.createNewFile();
                FileOutputStream fOut = new FileOutputStream(myFile);
                OutputStreamWriter myOutWriter =
                                                        new
OutputStreamWriter(fOut);
                myOutWriter.append(txtData.getText());
                myOutWriter.close();
                fOut.close();
                Toast.makeText(getBaseContext(),"Done writing SD
                'mysdfile.txt'",Toast.LENGTH_SHORT).show();
        } catch (Exception e) {
                Toast.makeText(getBaseContext(), e.getMessage(),
                                Toast.LENGTH_SHORT).show();
        }
}// onClick
}); // btnWriteSDFile
        btnReadSDFile = (Button) findViewById(R.id.btnReadSDFile);
        btnReadSDFile.setOnClickListener(new OnClickListener() {

        public void onClick(View v) {
                // write on SD card file data in the text box
        try {
                File myFile = new File("/sdcard/mysdfile.txt");
                FileInputStream fIn = new FileInputStream(myFile);
                BufferedReader myReader = new BufferedReader(
```

```
                        new InputStreamReader(fIn));
                String aDataRow = "";
                String aBuffer = "";
                while ((aDataRow = myReader.readLine()) != null) {
                        aBuffer += aDataRow + "\n";
                }
                txtData.setText(aBuffer);
                myReader.close();
                Toast.makeText(getBaseContext(),"Done reading SD
                'mysdfile.txt'",Toast.LENGTH_SHORT).show();
        } catch (Exception e) {
                Toast.makeText(getBaseContext(), e.getMessage(),
                        Toast.LENGTH_SHORT).show();
        }
        }// onClick
        }); // btnReadSDFile

        btnClearScreen = (Button) findViewById(R.id.btnClearScreen);
        btnClearScreen.setOnClickListener(new OnClickListener() {
                public void onClick(View v) {
                        // clear text box
                        txtData.setText("");
                }
        }); // btnClearScreen
        btnClose = (Button) findViewById(R.id.btnClose);
        btnClose.setOnClickListener(new OnClickListener() {

                public void onClick(View v) {
                        // clear text box
                        finish();
                }
        }); // btnClose
    }// onCreate
}// AndSDcard
```

main.xml

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

```
<LinearLayout
android:layout_width="fill_parent"
android:layout_height="fill_parent"
android:background="#ff0000ff"
android:orientation="vertical"
xmlns:android="http://schemas.android.com/apk/res/android"
>
        <EditText
        android:id="@+id/txtData"
        android:layout_width="fill_parent"
        android:layout_height="180px"
        android:textSize="18sp" />
        <Button
        android:id="@+id/btnWriteSDFile"
        android:layout_width="143px"
        android:layout_height="44px"
        android:text="1. Write SD File" />
        <Button
        android:id="@+id/btnClearScreen"
        android:layout_width="141px"
        android:layout_height="42px"
        android:text="2. Clear Screen" />
        <Button
        android:id="@+id/btnReadSDFile"
        android:layout_width="140px"
        android:layout_height="42px"
        android:text="3. Read SD File" />
        <Button
        android:id="@+id/btnClose"
        android:layout_width="141px"
        android:layout_height="43px"
        android:text="4. Close" />

</LinearLayout>
```

SQLiteDatabase For Android

Android applications can store application data in SQLite databases. In this tutorial, you learn how SQLite databases are designed and manipulated.

Here we begin by designing and using a simple SQLite database to manage chess tournament scores. This tutorial is meant as a brief overview of how SQLite databases work. This knowledge will then be used in future development tutorials to implement database-driven Android applications.

Getting Started

Android applications can create and manipulate their own private SQLite relational databases. Developers can also inspect and modify databases on a given Android emulator or device using the sqlite3 command-line tool provided as part of the Android SDK tool called Android Debug Bridge (adb).

In this we assume that you have some understanding of relational databases, in theory, but require a bit of a refresher course before you use them within your Android applications. This particular tutorial does not require any tools; it's more a theoretical exercise.

However, if you are planning to develop Android applications which rely upon SQLite databases, you will need to install the tools necessary for Android development, such as the Android SDK and the Eclipse IDE.

What is SQLite?

SQLite is a lightweight relational database engine. SQLite is fast and has a small footprint, making it perfect for Android devices. Instead of the heavyweight server-based databases like Oracle and Microsoft SQL Server, each SQLite database is stored within a single file on disk. Android applications can choose to store private application data in a SQLite database.

Note: If you're familiar with SQL, then SQLite will be very easy to pick up. SQLite is basically a stripped-down SQL database engine for embedded devices.

A Quick Review of Database Fundamentals

A database is simply a structured way of storing data in a persistent fashion. Data is stored in tables. A table has columns with different datatypes. Each row in a table represents a data record. You may find it helpful to think of a table like an Excel spreadsheet. For an object oriented programming perspective, each table in a database often represents an object (represented by a class). Each table column represents a class attribute. Each record in a table represents a specific instance of that object.

Let's look at a quick example. Let's say you have a company database with a table called Employee. The Employee table might have five typed columns: EmployeeID (number), FirstName (string), LastName (string), Title (string) and Salary (number). You could then add a record to the data base for an employee named John Doe and a separate record for an employee named Anne Droid.

Data within a database is meant to be inspected and manipulated. Data within a table can be:

- Added (using the INSERT command)
- Modified (using the UPDATE command)
- Removed (using the DELETE command)

You can search for specific data within a database using what is called a query. A query (using the SELECT command) may involve one table, or multiple tables. To create a query, you must specify the tables, data columns, and data values of interest using SQL command language. Each SQL command is terminated with a semicolon (;).

The Chess Tournament Database

The best way to truly understand how SQLite databases function is to work through a simple example, so let's do so. Let's pretend that we have an application that keeps track of player scores from a casual chess tournament. Player scores are recorded and then, at the end of a series of matches, the winner is determined. Each player's overall tournament score is calculated from their performance on:

- Four warm-up heats (weight: 10% of overall score each)
- One semi-final (weight: 25% of overall score)
- One final (weight: 35% of overall score)

Note: For the tournament, player scores could be based upon a formula that factors in the time it took to win the game and the type and number of pieces left on the board at the end of the game. This way, a strong player will receive a high score for losing few powerful pieces and winning the game quickly. Perhaps style and attitude are included by the judges to encourage fun, light play. How scores are calculated is really not important to how we define our database; we just store them in the database. For simplicity, we will assume that scores are based on a scale of 0–100.

Designing the Database Schema

A database schema is simply the definition of the structure of the database in terms of tables, data columns and such. The schema for our tournament database is quite simple:
The TournamentScores database schema has three tables:

- The Players table contains player information.
- The Games table contains information about each game and how much it counts toward the player's overall tournament score.
- The GameResults table contains all players' game scores.

SQLite3 has support for the following common datatypes for columns:

- INTEGER (signed integers)
- REAL (floating point values)
- TEXT (UTF-8 or UTF-16 string; encoded using database encoding)
- BLOB (data chunk)

Once you've determined which columns are necessary for each table, you're ready to create some tables within your database schema.

Working with Tables

Let's begin by creating the Players table. This table requires a unique player id to reference each player. We can make this the primary key (to uniquely identify a record in this table) and set its autoincrement attribute. Autoincrement means that each time a new player record is added, the record will get a new, unique player id. We also want to store the first and last name of each

player-no nulls allowed.
Here we can use the CREATE TABLE SQL statement to generate the Players table:

```
CREATE TABLE Players (
id INTEGER PRIMARY KEY AUTOINCREMENT,
fname TEXT NOT NULL,
lname TEXT NOT NULL );
```

The Games table is very similar. We need a unique game id to reference each game. We also want a friendly name for each game and a weight value for how much the game counts towards the player's final tournament score (as a percentage). Here's the SQL statement to create the Games table:

```
CREATE TABLE Games (
id INTEGER PRIMARY KEY AUTOINCREMENT,
gamename TEXT,
weight REAL DEFAULT .10 CHECK (weight<=1));
```

You can also delete tables using the DROP TABLE statement. For example, to delete the Games table, use the following SQL command:

```
DROP TABLE Games;
```

Populating Tables with Data Records

Before we move on, let's add some data to these tables. To add a record to the Players table, you need to specify the column names and the values in order. For example, the following SQL statement uses the INSERT command to add a record for chess player Bobby Fisher:

```
INSERT into Players
(fname, lname)
VALUES
('Bobby', 'Fisher');
```

While we're add it, we'll add two more players: Bart Simpson (a very pitiful chess player) and Garry Kasparov (perhaps the best chess player ever). At the same time, we need to add a bunch of records to the Games table. First we add the semi-final, which counts for 25 percent of the player's tournament score:

```
INSERT into Games
(gamename, weight)
VALUES
('Semi-Final', .25);
```

Then we add a couple warm-up heats, which use the default weight of 10 percent:

```
INSERT into Games (gamename) VALUES ('Warm-up Heat 1');
```

Finally, we add a final worth 35 percent of the total tournament score:

```
INSERT into Games
(gamename, weight)
VALUES
('Final', .35);
```

Querying Tables for Results with SELECT

How do we know the data we've added is in the table? Well, that's easy. We simply query for all rows in a table using a SELECT statement:

```
SELECT * FROM Games;
```

This returns all records in the Games table:

id	gamename	weight
1	Semi-Final	0.25
2	Warm-up Heat 1	0.1
3	Warm-up Heat 2	0.1

Using Column Aliases and Calculated Columns

We can also create our own columns and alias them. For example, we can create a column alias called PlayerName that is a calculated column: It's the player's first and last names concatenated using the || operator, separated by a space:

```
SELECT fname|| '||lname AS PlayerName, id FROM Players;
```

This query produces the following results:

PlayerName	id
Bobby Fisher	1
Bart Simpsen	2
Garry Kasparov	3

Altering Data in Tables

Bart's (player id 2) last name is spelled incorrectly. To update the Players table to reflect the correct spelling, you can use the UPDATE command:

```
UPDATE Players
SET lname='Simpson'
WHERE playerid=2;
```

You can delete rows from a table using the DELETE function. For example, to delete the record we just updated:

```
DELETE FROM Players WHERE playerid=2;
```

You can delete all rows in a table by not specifying the WHERE clause:

```
DELETE FROM Players;
```

Using Foreign Keys and Composite Keys

Now that we have our Players and Games all set up, let's create the GameResults table. This is a more complicated table. The GameResults table pairs up player ids from the Players table with game ids from the Games table and then lists the score that the player earned for the specific game. Columns, which link to other tables in this way, are often called foreign keys. We want unique player-game pairings, so we create a composite primary key from the player and game foreign keys, to uniquely identify a GameResults record. Lastly, we enforce that the scores are whole numbers between 0 and 100.

```
CREATE TABLE GameResults (
playerid INTEGER REFERENCES Players(id),
gameid INTEGER REFERENCES Games(id),
score INTEGER CHECK (score<=100 AND score>=0),
PRIMARY KEY (playerid, gameid));
```

(Note: SQLite does not enforce foreign key constraints, but you can set them up anyway and enforce the constraints by creating triggers.)

Now it's time to insert some data to the GameResults table. Let's say Bobby Fisher (player id 1) received a score of 82 points on the semi-final (game id 1). You could use the following SQL command to insert the appropriate record into the GameResults table:

```
INSERT into GameResults
(playerid, gameid, score)
VALUES
(1,1,82);
```

Now let's assume the tournament is played and the scores are added to the GameResults table. Bobby is a good player, Bart is a terrible player, and Garry always plays a perfect game. Once the records have been added to the GameResults table, we can perform a SELECT * command to list all records in the table, or we can specify columns explicitly like this:

```
SELECT playerid, gameid, score FROM GameResults;
```

Here are the results from this query:

playerid	gameid	score
1	1	82
1	2	88
1	3	78
1	4	90
1	5	85
1	6	94
2	1	10
2	2	60
2	3	50
2	4	55
2	5	45
3	5	100
3	4	100
3	3	100
3	2	100
3	1	100

As you can see, this listing is not particularly "human-readable".

Querying Across Multiple Tables Using JOIN

Wouldn't it be more helpful if the names of the players and Games were shown instead of their numeric ids? Querying and combining data in SELECT statements is often handled by performing a JOIN with multiple table sources; there are different kinds of JOINS. When you work with multiple tables, you need to specify which table that a column belongs to (especially when columns are named the same, such as with all these different id columns). You can refer to columns by their column name or by their table name, then a dot (.), and then the column name. Let's relist the player scores again, only this time, include the name of the game and the name of the player. Also, we limit our results only to the score for the Final (game id 6):

```
SELECT  Players.fname||' '|| Players.lname AS PlayerName,
Games.gamename,
GameResults.score
FROM GameResults
```

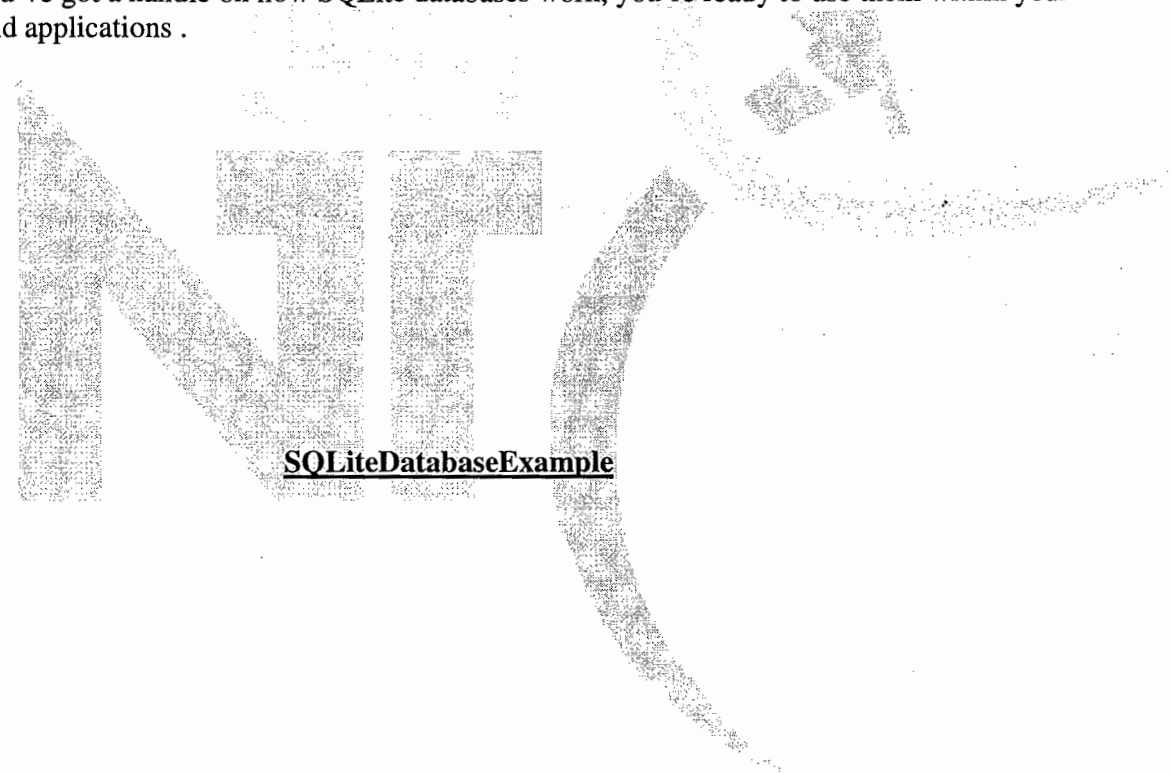
```
JOIN Players
    ON (GameResults.playerid=Players.id)
JOIN Games
    ON (GameResults.gameid=Games.id)
WHERE gameid=6;
```

which gives us the following results (you could leave off the WHERE to get all Games):

PlayerName	gamename	score
Bobby Fisher	Final	94
Bart Simpson	Final	65
Garry Kasparov	Final	100

Conclusion

This concludes our exploration of a simple SQLite database example: a chess tournament database. Hopefully you've reacquainted yourself with relational database concepts like tables, records, and queries and familiarized yourself with many of the commonly used SQLite commands. Finally, you've walked through the design and usage of an example database. Now that you've got a handle on how SQLite databases work, you're ready to use them within your Android applications .



SQLiteDatabaseExample

MyDbHelper.java

```
package com.rajender.database;

import android.content.Context;
import android.database.sqlite.SQLiteDatabase;
import android.database.sqlite.SQLiteOpenHelper;

public class MyDBHelper extends SQLiteOpenHelper{
        public static final String DATABASE
                                                    ="StudentDb.db";
        public static final int VERSION=1;
        public static final String ID="ID";
        public static final String NAME="NAME";
        public static final String PHONE="PHONE";
        public static final String ADDR="ADDR";
        public static final String TABLE="student";

        public MyDBHelper(Context ctx){
                super(ctx, DATABASE, null, VERSION);
        }
        @Override
        public void onCreate(SQLiteDatabase db) {
```

```
            db.execSQL("create table "+TABLE+"("+ID+" integer primary key
autoincrement," + NAME+" text,"+PHONE+" text,"+ADDR+" text)");
        }
        @Override
        public void onUpgrade(SQLiteDatabase db,
                int oldVersion, int newVersion) {
        }
}
```

main.xml

```
<?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"
   android:background="#F5F5DC">

   <TextView
      android:layout_width="fill_parent"
      android:layout_height="wrap_content"
      android:text="Student Registration"
      android:textColor="#f00"
      android:textSize="25px"
      android:textStyle="bold" />

   <TableRow
      android:layout_width="match_parent"
      android:layout_height="wrap_content" >

      <TextView
         android:layout_width="120px"
         android:layout_height="wrap_content"
         android:text="Name"
         android:textColor="#00f"
         android:textSize="20px" />

      <EditText
         android:id="@+id/nameText"
         android:layout_width="fill_parent"
         android:layout_height="wrap_content" />
   </TableRow>
   <TableRow
      android:layout_width="match_parent"
      android:layout_height="wrap_content" >

      <TextView
```

```
        android:layout_width="120px"
        android:layout_height="wrap_content"
        android:text="Phone"
        android:textColor="#00f"
        android:textSize="20px" />

    <EditText
        android:id="@+id/phoneText"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content" />
</TableRow>
<TableRow
    android:layout_width="match_parent"
    android:layout_height="wrap_content">

    <TextView
        android:layout_width="120px"
        android:layout_height="wrap_content"
        android:text="Address"
        android:textColor="#00f"
        android:textSize="20px" />

    <EditText
        android:id="@+id/addrText"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content" />
</TableRow>
 <TableRow
    android:layout_width="match_parent"
    android:layout_height="wrap_content">

    <Button
        android:onClick="saveFunction"
        android:layout_width="105px"
        android:layout_height="wrap_content"
        android:text="Save" />
    <Button
        android:onClick="selectFunction"
        android:layout_width="105px"
        android:layout_height="wrap_content"
        android:text="Select" />
     <Button
        android:onClick="resetFunction"
        android:layout_width="105px"
        android:layout_height="wrap_content"
        android:text="Reset" />
```

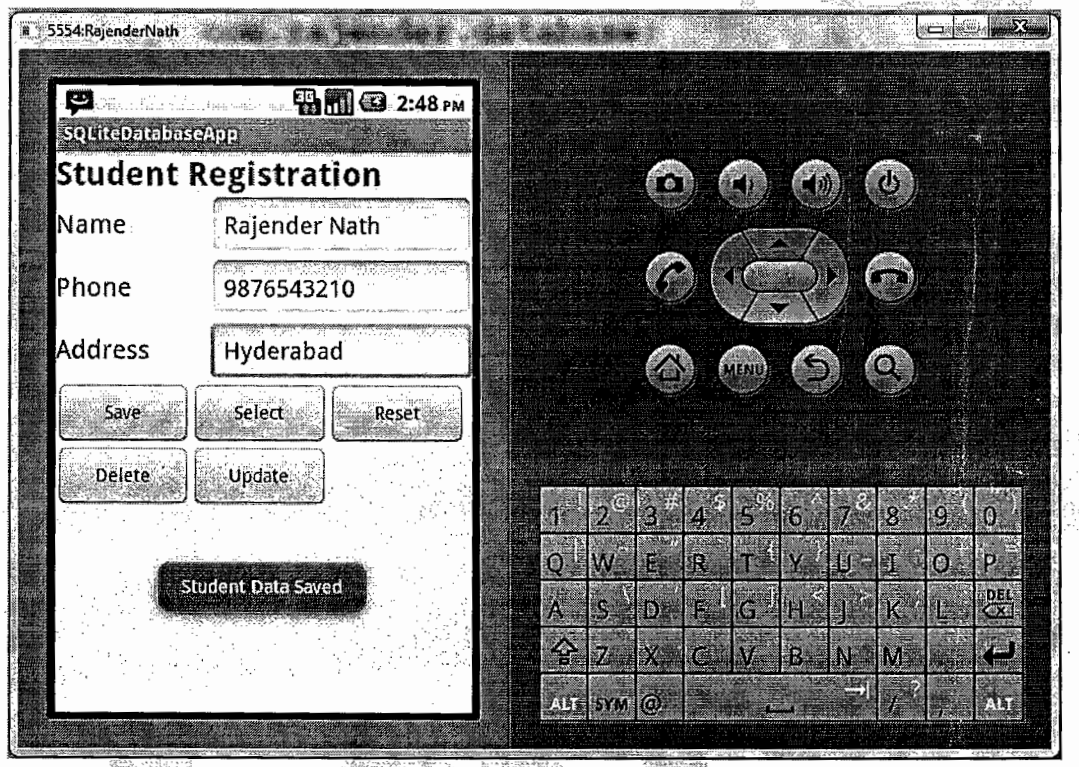
```
    </TableRow>

     <TableRow
       android:layout_width="match_parent"
       android:layout_height="wrap_content" >

        <Button
          android:onClick="deleteFunction"
          android:layout_width="105px"
          android:layout_height="wrap_content"
          android:text="Delete" />
        <Button
          android:onClick="updateFunction"
          android:layout_width="105px"
          android:layout_height="wrap_content"
          android:text="Update" />

    </TableRow>

</LinearLayout>
```

StudentDatabaseActivity.java

```
package com.rajender.database;

import android.app.Activity;
import android.content.Intent;
import android.database.sqlite.SQLiteDatabase;
import android.database.sqlite.SQLiteStatement;
import android.os.Bundle;
import android.view.View;
import android.widget.EditText;
import android.widget.Toast;

public class StudentDatabaseActivity extends Activity{
   MyDBHelper helper;
   SQLiteDatabase db;
   SQLiteStatement st;
   EditText nameET,phoneET,addrET;
   String name,phone,addr;

        @Override
   public void onCreate(Bundle b) {
```

```
super.onCreate(b);
setContentView(R.layout.main);

nameET=(EditText)findViewById(R.id.nameText);
phoneET=(EditText)findViewById(R.id.phoneText);
addrET=(EditText)findViewById(R.id.addrText);

//Create database
helper = new MyDBHelper(this);
//open database connection
db = helper.getWritableDatabase();
}

    public void saveFunction(View v){

        name=nameET.getText().toString().trim();
        phone=phoneET.getText().toString().trim();
        addr=addrET.getText().toString().trim();
        try {
                st = db.compileStatement(
                        "insert into student values(?,?,?,?)");
                st.bindString(2, name);
                st.bindString(3, phone);
                st.bindString(4, addr);
                st.executeInsert();
                Toast.makeText(getApplicationContext(),
                                "Student Data Saved",
                                Toast.LENGTH_LONG).show();
        } catch (Exception e) {
                Toast.makeText(getApplicationContext(),
                "Student Data Not Saved"+e.getMessage(),
                Toast.LENGTH_LONG).show();
        }
    }
    //Clear EditText Values
    public void resetFunction(View v){
            nameET.setText("");
            phoneET.setText("");
            addrET.setText("");
    }
    //select data from DisplayActivity
    public void selectFunction(View v){
            startActivity(new Intent(this,
                                DisplayActivity.class));
    }
    public void deleteFunction(View v){
```

```
        startActivity(new Intent(this,DeleteActivity.class));
    }

    public void updateFunction(View v){
        startActivity(new Intent(this,UpdateActivity.class));
    }

}
```

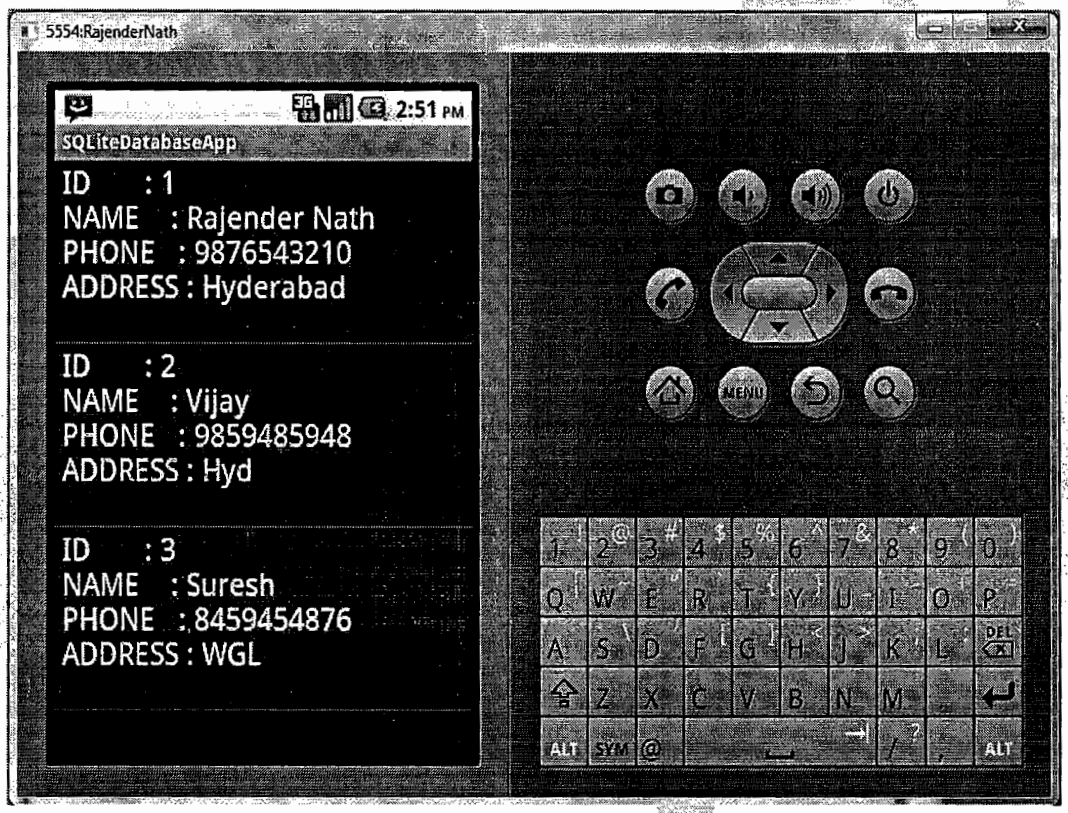


DisplayActivity .java

```
package com.rajender.database;

import java.util.ArrayList;

import android.app.ListActivity;
import android.database.Cursor;
import android.database.sqlite.SQLiteDatabase;
import android.os.Bundle;
import android.widget.ArrayAdapter;
```

```
public class DisplayActivity extends ListActivity{
        MyDBHelper helper;
        SQLiteDatabase db;
        Cursor cursor;
        ArrayList<String> al;
        @Override
        protected void onCreate(Bundle b) {
                super.onCreate(b);
                al=new ArrayList<String>();

                helper=new MyDBHelper(this);
                db=helper.getReadableDatabase();

                try {
                        cursor = db.rawQuery("select * from student",  null);
                        if(cursor != null){
                                if(cursor.moveToFirst()){
                                        do{
                                                String id = cursor.getString(
                                                        cursor.getColumnIndex("ID"));
                                                String name = cursor.getString(
                                                        cursor.getColumnIndex("NAME"));
                                                String phone = cursor.getString(
                                                        cursor.getColumnIndex("PHONE"));
                                                String addr = cursor.getString(
                                                        cursor.getColumnIndex("ADDR"));

                                                //Store database values into ArrayList
                                                al.add("ID      :"+id+"\n"+
                                                        "NAME    : "+name+"\n"+
                                                        "PHONE   : "+phone+"\n"+
                                                        "ADDRESS : "+addr+"\n");
                                        }
                                        while(cursor.moveToNext());
                                }
                        }
```

```
            } catch (Exception e) {
            }
            ArrayAdapter<String> adapter=
                    new ArrayAdapter<String>(this,
                    android.R.layout.simple_list_item_1,al);
            setListAdapter(adapter);
        }
    }
```

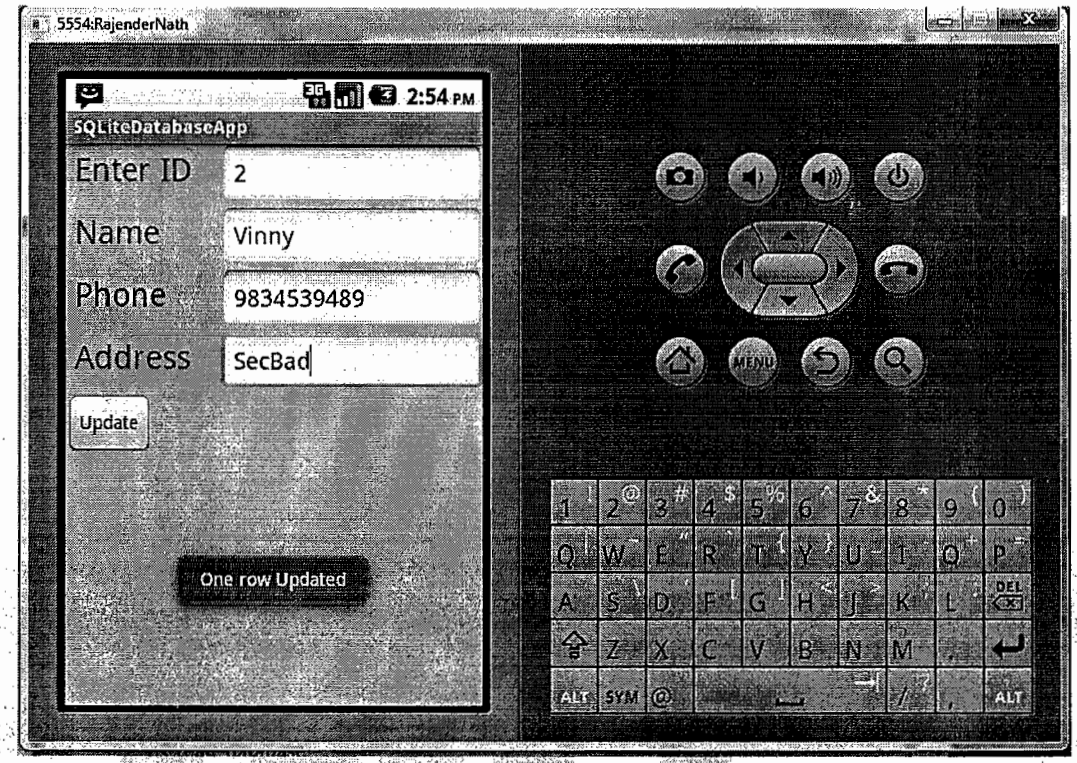


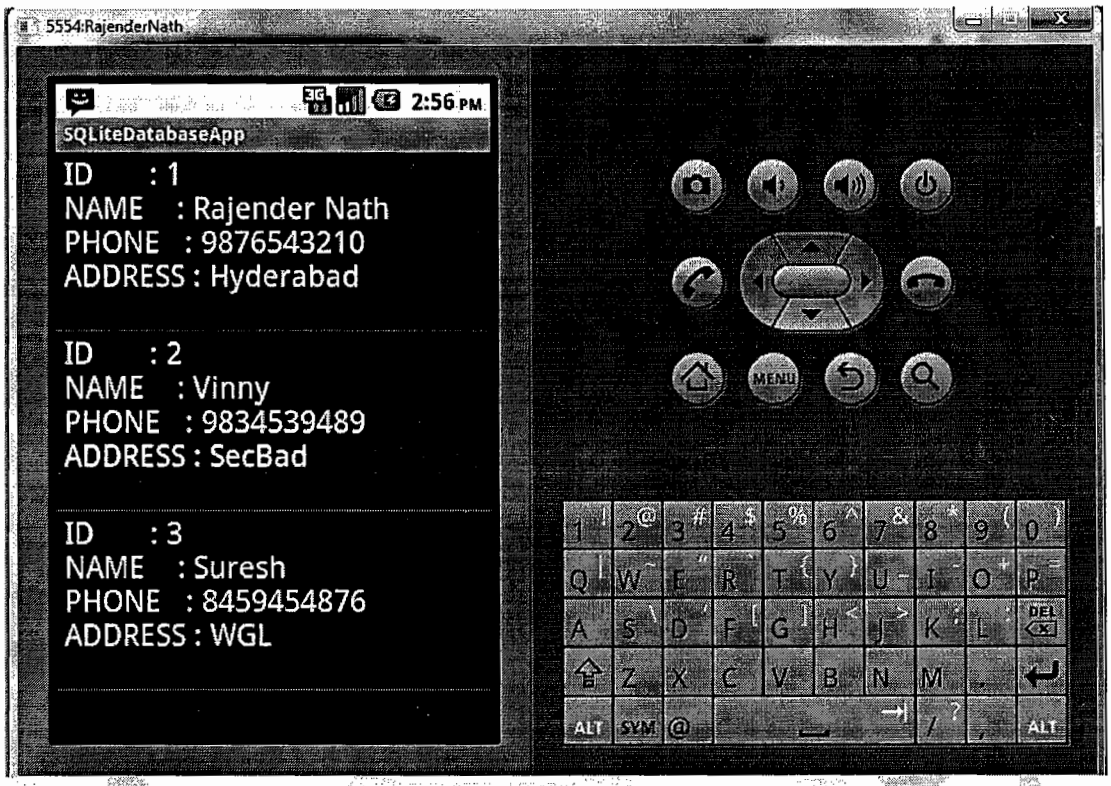
UpdateActivity.java

```
package com.rajender.database;

import android.app.Activity;
import android.content.ContentValues;
import android.database.sqlite.SQLiteDatabase;
import android.os.Bundle;
import android.view.View;
import android.widget.EditText;
import android.widget.Toast;
```

```
public class UpdateActivity extends Activity {
        MyDBHelper helper;
   SQLiteDatabase db;
   EditText idText,nameET,phoneET,addrET;
   String id,name,phone,addr;
   @Override
   public void onCreate(Bundle b) {
      super.onCreate(b);
      setContentView(R.layout.update);

      idText=(EditText)findViewById(R.id.idText);
      nameET=(EditText)findViewById(R.id.nameText);
      phoneET=(EditText)findViewById(R.id.phoneText);
      addrET=(EditText)findViewById(R.id.addrText);

      //Create database
      helper = new MyDBHelper(this);
      //open database connection
      db = helper.getWritableDatabase();
      }

        public void updateFunction(View v){
              id=idText.getText().toString().trim();
              name=nameET.getText().toString().trim();
             phone=phoneET.getText().toString().trim();
             addr=addrET.getText().toString().trim();

             ContentValues cv=new ContentValues();
                   cv.put(MyDBHelper.NAME, name);
                   cv.put(MyDBHelper.PHONE, phone);
                   cv.put(MyDBHelper.ADDR, addr);

          db.update(MyDBHelper.TABLE, cv, "ID = ?", new String[] { id });

          Toast.makeText(getApplicationContext(),
```

```
                "One row Updated",
                Toast.LENGTH_LONG).show();
    }
}
```

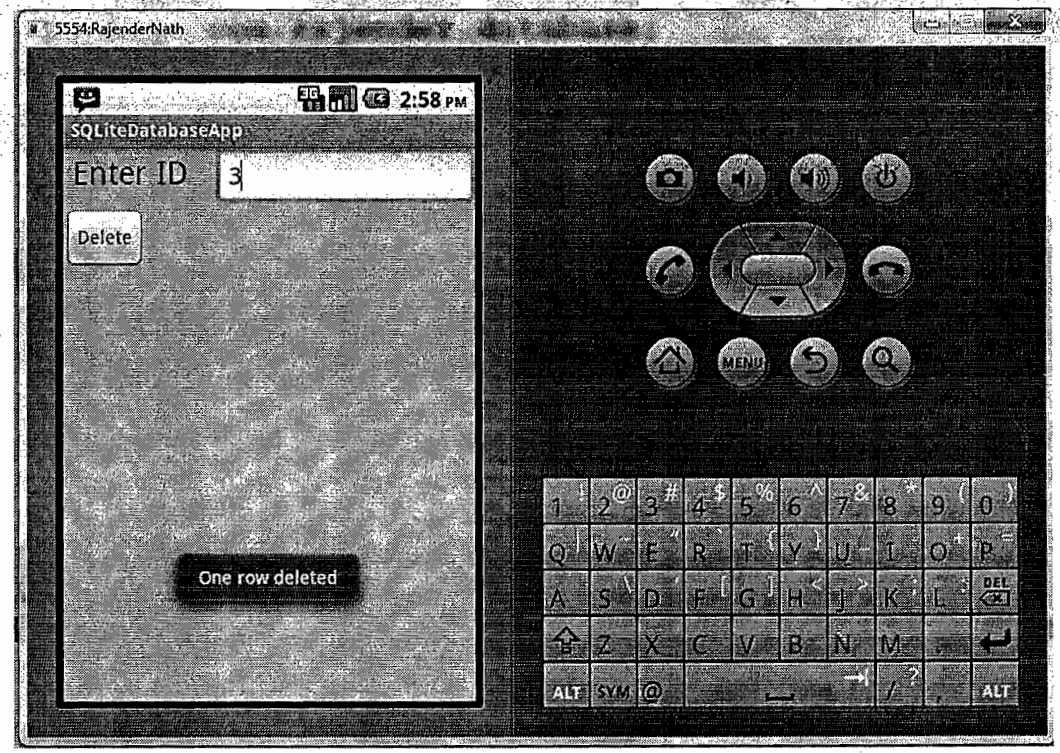


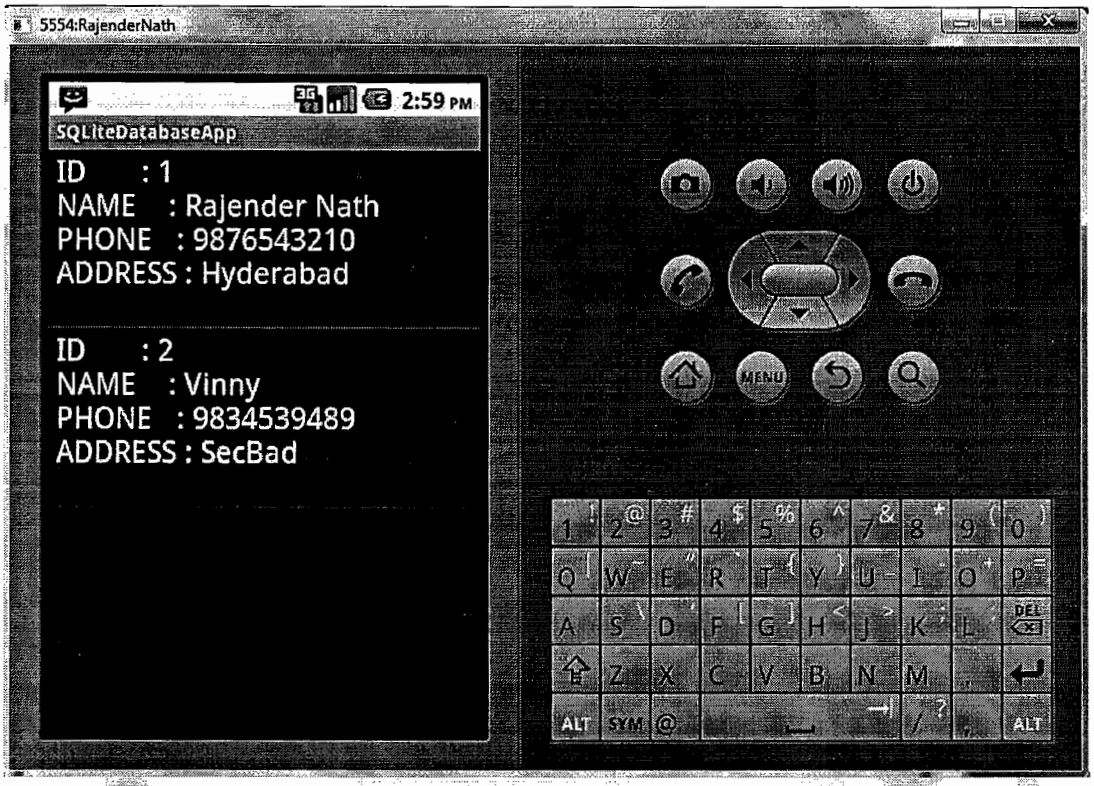


DeleteActivity.java

```
package com.rajender.database;

import android.app.Activity;
import android.database.sqlite.SQLiteDatabase;
import android.os.Bundle;
import android.view.View;
import android.widget.EditText;
import android.widget.Toast;

public class DeleteActivity extends Activity {
        MyDBHelper helper;
        SQLiteDatabase db;
        EditText idText;
        String id;
        @Override
        public void onCreate(Bundle b) {
          super.onCreate(b);
```

```
setContentView(R.layout.delete);
idText=(EditText)findViewById(R.id.idText);

//Create database
helper = new MyDBHelper(this);
//open database connection
db = helper.getWritableDatabase();
}

public void deleteFunction(View v){
        id=idText.getText().toString().trim();

    db.execSQL("delete from student where ID="+id);
    Toast.makeText(getApplicationContext(),
                "One row deleted",
                Toast.LENGTH_LONG).show();
}
}
```





AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
   package="com.rajender.database"
   android:versionCode="1" android:versionName="1.0" >

   <uses-sdk android:minSdkVersion="8" />
   <application
      android:icon="@drawable/ic_launcher"
      android:label="@string/app_name" >
      <activity
                android:name=".StudentDatabaseActivity"
                 android:label="@string/app_name" >
         <intent-filter>
            <action android:name="android.intent.action.MAIN" />
            <category android:name="android.intent.category.LAUNCHER" />
         </intent-filter>
```

```
    </activity>
    <activity android:name="DisplayActivity"/>
    <activity android:name="DeleteActivity"/>
    <activity android:name="UpdateActivity"/>
  </application>

</manifest>
```

Animation

An animation resource can define one of two types of animations:

Property Animation

Creates an animation by modifying an object's property values over a set period of time with an Animator.

View Animation

There are two types of animations that you can do with the view animation framework:

- Tween animation: Creates an animation by performing a series of transformations on a single image with an Animation
- Frame animation: or creates an animation by showing a sequence of images in order with an AnimationDrawable.

Tween animation

An animation defined in XML that performs transitions such as rotating, fading, moving, and stretching on a graphic.

FILE LOCATION:

`res/anim/`*filename*`.xml`

The filename will be used as the resource ID.

COMPILED RESOURCE DATATYPE:

Resource pointer to an Animation.

RESOURCE REFERENCE:

In Java: `R.anim.`*filename*

In XML: `@[`*package*`:]anim/`*filename*

SYNTAX:

```
<?xml version="1.0" encoding="utf-8"?>
<set xmlns:android="http://schemas.android.com/apk/res/android"
   android:interpolator="@[package:]anim/interpolator_resource"
   android:shareInterpolator=["true" | "false"] >
   <alpha
      android:fromAlpha="float"
      android:toAlpha="float" />
   <scale
      android:fromXScale="float"
      android:toXScale="float"
      android:fromYScale="float"
      android:toYScale="float"
      android:pivotX="float"
      android:pivotY="float" />
   <translate
      android:fromXDelta="float"
      android:toXDelta="float"
      android:fromYDelta="float"
      android:toYDelta="float" />
   <rotate
      android:fromDegrees="float"
      android:toDegrees="float"
      android:pivotX="float"
      android:pivotY="float" />
   <set>
      ...
   </set>
</set>
```

The file must have a single root element: either an`<alpha>`, `<scale>`, `<translate>`,`<rotate>`, or `<set>` element that holds a group (or groups) of other animation elements (even nested `<set>` elements).

ELEMENTS:

<set>

A container that holds other animation elements (`<alpha>`, `<scale>`, `<translate>`,`<rotate>`) or other `<set>` elements. Represents an AnimationSet.

ATTRIBUTES:

android:interpolator

Interpolator resource. An Interpolator to apply on the animation. The value must be a reference to a resource that specifies an interpolator (not an interpolator class name). There are default interpolator resources available from the platform or you can create your own interpolator resource. See the discussion below for more about Interpolators.

android:shareInterpolator
Boolean. "true" if you want to share the same interpolator among all child elements.

<alpha>
A fade-in or fade-out animation. Represents an AlphaAnimation.

ATTRIBUTES:
android:fromAlpha
Float. Starting opacity offset, where 0.0 is transparent and 1.0 is opaque.

android:toAlpha
Float. Ending opacity offset, where 0.0 is transparent and 1.0 is opaque.

For more attributes supported by `<alpha>`, see theAnimation class reference (of which, all XML attributes are inherrited by this element).
<scale>
A resizing animation. You can specify the center point of the image from which it grows outward (or inward) by specifying `pivotX` and `pivotY`. For example, if these values are 0, 0 (top-left corner), all growth will be down and to the right. Represents a ScaleAnimation.

ATTRIBUTES:
android:fromXScale
Float. Starting X size offset, where 1.0 is no change.

android:toXScale
Float. Ending X size offset, where 1.0 is no change.

android:fromYScale
Float. Starting Y size offset, where 1.0 is no change.

android:toYScale
Float. Ending Y size offset, where 1.0 is no change.

android:pivotX
Float. The X coordinate to remain fixed when the object is scaled.

android:pivotY
Float. The Y coordinate to remain fixed when the object is scaled.

For more attributes supported by `<scale>`, see theAnimation class reference (of which, all XML attributes are inherrited by this element).
<translate>

A vertical and/or horizontal motion. Supports the following attributes in any of the following three formats: values from -100 to 100 ending with "%", indicating a percentage relative to itself; values from -100 to 100 ending in "%p", indicating a percentage relative to its parent; a float value with no suffix, indicating an absolute value. Represents a TranslateAnimation.

ATTRIBUTES:

android:fromXDelta

Float or percentage. Starting X offset. Expressed either: in pixels relative to the normal position (such as `"5"`), in percentage relative to the element width (such as`"5%"`), or in percentage relative to the parent width (such as `"5%p"`).

android:toXDelta

Float or percentage. Ending X offset. Expressed either: in pixels relative to the normal position (such as `"5"`), in percentage relative to the element width (such as`"5%"`), or in percentage relative to the parent width (such as `"5%p"`).

android:fromYDelta

Float or percentage. Starting Y offset. Expressed either: in pixels relative to the normal position (such as `"5"`), in percentage relative to the element height (such as`"5%"`), or in percentage relative to the parent height (such as `"5%p"`).

android:toYDelta

Float or percentage. Ending Y offset. Expressed either: in pixels relative to the normal position (such as `"5"`), in percentage relative to the element height (such as`"5%"`), or in percentage relative to the parent height (such as `"5%p"`).

For more attributes supported by `<translate>`, see theAnimation class reference (of which, all XML attributes are inherrited by this element).

<rotate>

A rotation animation. Represents a RotateAnimation.

ATTRIBUTES:

android:fromDegrees

Float. Starting angular position, in degrees.

android:toDegrees

Float. Ending angular position, in degrees.

android:pivotX

Float or percentage. The X coordinate of the center of rotation. Expressed either: in pixels relative to the object's left edge (such as `"5"`), in percentage relative to the object's

left edge (such as "5%"), or in percentage relative to the parent container's left edge (such as "5%p").

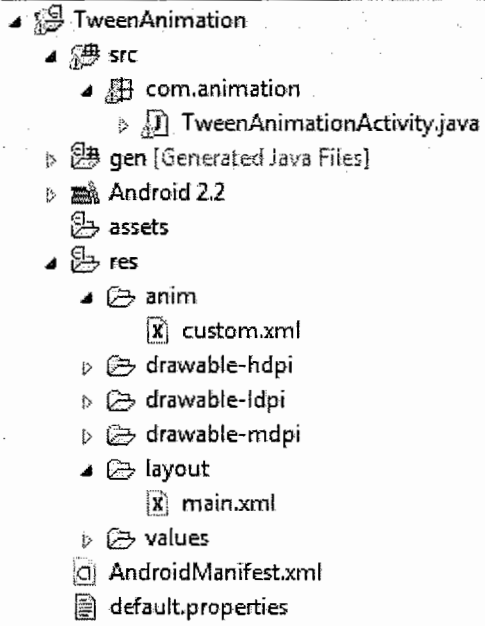
android:pivotY
Float or percentage. The Y coordinate of the center of rotation. Expressed either: in pixels relative to the object's top edge (such as "5"), in percentage relative to the object's top edge (such as "5%"), or in percentage relative to the parent container's top edge (such as "5%p").

For more attributes supported by <rotate>, see theAnimation class reference (of which, all XML attributes are inherrited by this element).

This application code will apply the animation to an ImageView and start the animation:

```
ImageView image = (ImageView) findViewById(R.id.image);
Animation hyperspaceJump = AnimationUtils.loadAnimation(this,
R.anim.hyperspace_jump);
image.startAnimation(hyperspaceJump);
```

TweenAnimation Example



main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
  android:orientation="vertical"
```

```
   android:layout_width="fill_parent"
   android:layout_height="fill_parent"
   >
<TextView
   android:layout_width="fill_parent"
   android:layout_height="wrap_content"
   android:text="Welcome"
   android:textColor="#00ff00"
   android:textSize="30px"
   android:id="@+id/textView"
   />
 <ImageView
        android:id="@+id/imageView"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:src="@drawable/imagesandroidrobo"/>
</LinearLayout>
```

custom.xml

```
<?xml version="1.0" encoding="utf-8"?>
<set
xmlns:android="http://schemas.android.com/apk/res/android"
android:shareInterpolator="false">

 <alpha
        android:fromAlpha="0.6"
        android:toAlpha="1.0"
        android:duration="4000">
</alpha> -->
<scale
        android:pivotX="150%"
        android:pivotY="250%"
        android:fromXScale=".1"
        android:fromYScale=".1"
        android:toXScale="1.0"
        android:toYScale="1.0"
        android:duration="20000" />
```

```
<rotate
        android:fromDegrees="0"
        android:toDegrees="720"
        android:pivotX="50%"
        android:pivotY="50%"
        android:duration="30000" />

</set>
```

TweenAnimationActivity.java

```
package com.animation;
import android.app.Activity;
import android.os.Bundle;
import android.view.animation.Animation;
import android.view.animation.AnimationUtils;
import android.widget.ImageView;
import android.widget.TextView;
public class TweenAnimationActivity extends Activity {
   @Override
   public void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.main);
      ImageView img=(ImageView)findViewById(R.id.imageView);
      TextView tv=(TextView)findViewById(R.id.textView);
      Animation animation=AnimationUtils.loadAnimation(this,R.anim.custom);
      //img.startAnimation(animation);
      tv.startAnimation(animation);
   }
}
```

Frame animation

An animation defined in XML that shows a sequence of images in order (like a film).

FILE LOCATION:

`res/drawable/`*filename*`.xml`

The filename will be used as the resource ID.

COMPILED RESOURCE DATATYPE:

Resource pointer to an AnimationDrawable.

RESOURCE REFERENCE:

In Java: `R.drawable.`*filename*

In XML: `@[`*package*`:]drawable.`*filename*

SYNTAX:

```
<?xml version="1.0" encoding="utf-8"?>
<animation-list xmlns:android="http://schemas.android.com/apk/res/android"
  android:oneshot=["true" | "false"] >
  <item
    android:drawable="@[package:]drawable/drawable_resource_name"
    android:duration="integer" />
</animation-list>
```

ELEMENTS:

<animation-list>

Required. This must be the root element. Contains one or more`<item>` elements.

ATTRIBUTES:

android:oneshot

Boolean. "true" if you want to perform the animation once; "false" to loop the animation.

<item>

A single frame of animation. Must be a child of a `<animation-list>` element.

ATTRIBUTES:

android:drawable

Drawable resource. The drawable to use for this frame.

android:duration

Integer. The duration to show this frame, in milliseconds.

EXAMPLE:

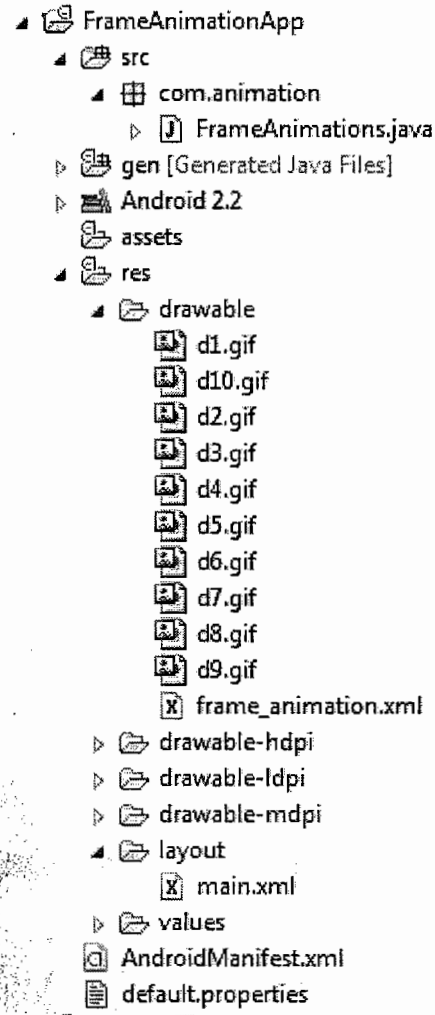
XML file saved at `res/anim/rocket.xml`:

```
<?xml version="1.0" encoding="utf-8"?>
<animation-list xmlns:android="http://schemas.android.com/apk/res/android"
   android:oneshot="false">
   <item android:drawable="@drawable/rocket_thrust1" android:duration="200" />
   <item android:drawable="@drawable/rocket_thrust2" android:duration="200" />
   <item android:drawable="@drawable/rocket_thrust3" android:duration="200" />
</animation-list>
```

This application code will set the animation as the background for a View, then play the animation:

```
ImageView rocketImage = (ImageView) findViewById(R.id.rocket_image);
rocketImage.setBackgroundResource(R.drawable.rocket_thrust);

rocketAnimation = (AnimationDrawable) rocketImage.getBackground();
rocketAnimation.start();
```

Frame Animation Example



FrameAnimations.java

```
package com.animation;

import android.app.Activity;
import android.graphics.drawable.AnimationDrawable;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.ImageView;

public class FrameAnimations extends Activity {
   @Override
   public void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.main);
```

```
        Button b = (Button)this.findViewById(R.id.startFAButtonId);
                b.setOnClickListener(new Button.OnClickListener() {
                 public void onClick(View v) {
                        ImageView imgView = (ImageView)findViewById(R.id.animationImage);
                       imgView.setBackgroundResource(R.drawable.frame_animation);
                    AnimationDrawable frameAnimation =
                                (AnimationDrawable) imgView.getBackground();
                       if (frameAnimation.isRunning()) {
                         frameAnimation.stop();
                       }
                       else {
                           frameAnimation.start();
                       }
                     }

                });
    }
}
```

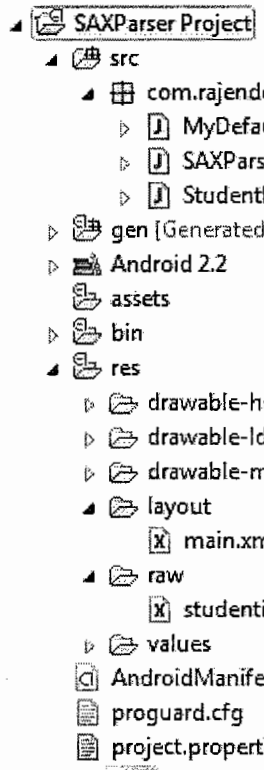
custom.xml

```
<animation-list
        xmlns:android="http://schemas.android.com/apk/res/android"
        android:oneshot="false">
  <item android:drawable="@drawable/d1" android:duration="250" />
  <item android:drawable="@drawable/d2" android:duration="150" />
  <item android:drawable="@drawable/d3" android:duration="250" />
  <item android:drawable="@drawable/d4" android:duration="450" />
  <item android:drawable="@drawable/d5" android:duration="50" />
  <item android:drawable="@drawable/d6" android:duration="50" />
  <item android:drawable="@drawable/d7" android:duration="250" />
  <item android:drawable="@drawable/d8" android:duration="0" />
  <item android:drawable="@drawable/d9" android:duration="150" />
  <item android:drawable="@drawable/d10" android:duration="150" />
</animation-list>
```

main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
  android:orientation="vertical"
  android:layout_width="fill_parent"
  android:layout_height="fill_parent">
    <TextView android:id="@+id/textViewId1"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="Frame By Frame Animation of 10 Dukes"/>
  <Button
    android:id="@+id/startFAButtonId"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="Start Duke Animation"
    />
  <ImageView
    android:id="@+id/animationImage"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"/>
</LinearLayout>
```

SAX Parser Project



studentinfo.xml

```
<students>
        <student>
                <sno>1001</sno>
                <sname>Ramesh</sname>
                <addr>Hyd</addr>
        </student>
        <student>
                <sno>1002</sno>
                <sname>Vijay</sname>
                <addr>Hyd</addr>
        </student>
        <student>
                <sno>1003</sno>
                <sname>Kiram</sname>
                <addr>SRNagar</addr>
        </student>
</students>
```

main.xml

```
<?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" >

 <ListView
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
      android:id="@+id/listView" />

</LinearLayout>
```

StudentPojo.java

```
package com.rajender.sax.parser;
import java.util.ArrayList;
public class StudentPojo {
        ArrayList<String> stdNo=new ArrayList<String>();
        ArrayList<String> stdName=new ArrayList<String>();
        ArrayList<String> address=new ArrayList<String>();

        public void setStdNo(String stdNo){
                this.stdNo.add(stdNo);
        }
        public ArrayList<String> getStdNo(){
                return stdNo;
        }
        public void setStdName(String stdName){
                this.stdName.add(stdName);
        }
        public ArrayList<String> getStdName(){
                return stdName;
        }
        public void setAddress(String address){
                this.address.add(address);
        }
        public ArrayList<String> getAddress(){
                return address;
        }
}
```

MyDefaultHandler.java

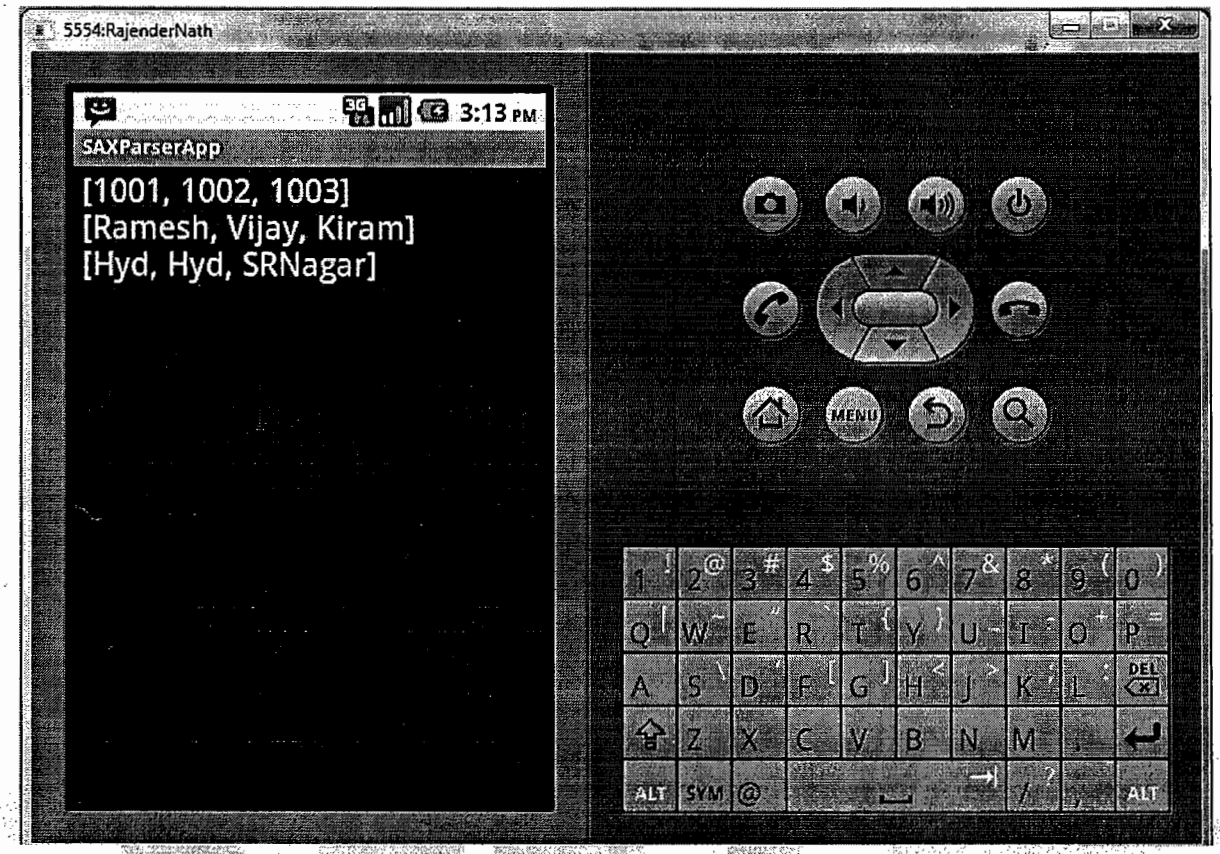
```
package com.rajender.sax.parser;

import org.xml.sax.Attributes;
import org.xml.sax.SAXException;
import org.xml.sax.helpers.DefaultHandler;

public class MyDefaultHandler extends DefaultHandler {

        static StudentPojo stdPojo;
        String str;
```

```
        @Override
        public void characters(char[] ch, int start, int length)
                        throws SAXException {
                super.characters(ch, start, length);
                str=new String(ch, start, length);
        }
        @Override
        public void startElement(String uri, String localName, String qName,
                        Attributes attributes) throws SAXException {
                super.startElement(uri, localName, qName, attributes);
                if(localName.equals("students")){
                        stdPojo=new StudentPojo();
                }
        }
        @Override
        public void endElement(String uri, String localName, String qName)
                        throws SAXException {
                super.endElement(uri, localName, qName);
                if(localName.equals("sno")){
                        stdPojo.setStdNo(str);
                }
                else if(localName.equals("sname")){
                        stdPojo.setStdName(str);
                }
                else if(localName.equals("addr")){
                        stdPojo.setAddress(str);
                }
        }
}
```

SAXParserActivity.java

```
package com.rajender.sax.parser;

import java.io.InputStream;
import java.util.ArrayList;
import javax.xml.parsers.SAXParser;
import javax.xml.parsers.SAXParserFactory;
import org.xml.sax.InputSource;
import org.xml.sax.XMLReader;
import android.app.Activity;
import android.os.Bundle;
import android.widget.ArrayAdapter;
import android.widget.ListView;

public class SAXParserActivity extends Activity {
   StudentPojo student;
   ArrayList<String> sno,sname,addr,al;
```

```
        @Override
    public void onCreate(Bundle b) {
        super.onCreate(b);
        setContentView(R.layout.main);

        sno=new ArrayList<String>();
        sname=new ArrayList<String>();
        addr=new ArrayList<String>();
        al=new ArrayList<String>();

        //Storing data
        InputStream is=getResources()
             .openRawResource(R.raw.studentinfo);
        try {
                                SAXParserFactory factory=SAXParserFactory.newInstance();
                                SAXParser saxParser=factory.newSAXParser();
                                XMLReader xmlReader=saxParser.getXMLReader();

                                xmlReader.setContentHandler(new MyDefaultHandler());
                                xmlReader.parse(new InputSource(is));
                        } catch (Exception e) {
                        }

        //Reading data
        student=MyDefaultHandler.stdPojo;

        sno= student.getStdNo();
        sname=student.getStdName();
        addr=student.getAddress();

        al.add(sno+"\n"+sname+"\n"+addr);

        ArrayAdapter<String> adapter=new ArrayAdapter<String>(this,
                        android.R.layout.simple_list_item_1,al);

        ListView list=(ListView)findViewById(R.id.listView);
        list.setAdapter(adapter);
        }
}
```



Sensors Project

- AcceleroMeterSensor Project
  - src
    - com.rajender.sensor
      - AcceleroMeterActivity.java
  - gen [Generated Java Files]
  - Android 2.2
  - JRE System Library [J2SE-1.5]
  - assets
  - bin
  - res
    - drawable-hdpi
    - drawable-ldpi
    - drawable-mdpi
    - layout
      - main.xml
    - values
  - AndroidManifest.xml
  - proguard.cfg
  - project.properties

main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
   android:layout_width="fill_parent"
   android:layout_height="fill_parent"
   android:orientation="vertical" >

   <TextView
      android:layout_width="fill_parent"
      android:layout_height="wrap_content"
      android:text="Sensor Application"
      android:textColor="#f00"
      android:textSize="30px"
      android:textStyle="bold" />

   <TextView
      android:id="@+id/textView1"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:textColor="#0f0"
      android:textSize="25px" />

   <TextView
      android:id="@+id/textView2"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
       android:textColor="#0f0"
      android:textSize="25px" />

   <TextView
      android:id="@+id/textView3"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
       android:textColor="#0f0"
      android:textSize="25px" />

</LinearLayout>
```

AcceleroMeterActivity.java

```
package com.rajender.sensor;

import android.app.Activity;
import android.content.Context;
import android.hardware.Sensor;
import android.hardware.SensorEvent;
import android.hardware.SensorEventListener;
```

```
import android.hardware.SensorManager;
import android.os.Bundle;
import android.widget.TextView;

public class AcceleroMeterActivity extends
                        Activity implements SensorEventListener{
   SensorManager sm;
   Sensor sensor;
   TextView xCoordinate,yCoordinate,zCoordinate;
        @Override
   public void onCreate(Bundle b) {
      super.onCreate(b);
      setContentView(R.layout.main);

      xCoordinate=(TextView)findViewById(R.id.textView1);
      yCoordinate=(TextView)findViewById(R.id.textView2);
      zCoordinate=(TextView)findViewById(R.id.textView3);

      sm=(SensorManager)getSystemService
                        (Context.SENSOR_SERVICE);
      sensor=sm.getDefaultSensor(
         SensorManager.SENSOR_ACCELEROMETER
     );
      sm.registerListener(this, sensor,
          SensorManager.SENSOR_DELAY_NORMAL);
   }
        public void onAccuracyChanged(
                        Sensor sensor, int accuracy) {

        }
        public void onSensorChanged(SensorEvent event){
                //Check Sensor type
                if(event.sensor.getType()==Sensor.TYPE_ACCELEROMETER){
                        float x=event.values[0];
                        float y=event.values[1];
                        float z=event.values[2];
```

```
                xCoordinate.setText("X  :  "+x);
                yCoordinate.setText("Y  :  "+y);
                zCoordinate.setText("Z  :  "+z);

            }
        }
    }
```