

**PROJECT REPORT**

(SEMESTER TRAINING)

**ANDROID APPLICATIONS**

SUBMITTED BY

**NAVJOT SINGH**

**ROLL NO. 10703066**

UNDER THE GUIDANCE OF

**Mr. Anil Vashisht**

Asst. Professor  
Department Of Computer  
Science and Engineering

**Mr. Vijay Solanki**

Chief Engineer  
Samsung India Software  
Engineering Lab, Noida

**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**

**THAPAR UNIVERSITY, PATIALA**

**(DEEMED UNIVERSITY)**

**JAN-MAY/JUN 2010**

## **DECLARATION**

I hereby declare that the project work entitled “Android Applications” is an authentic record of my own work carried out at Samsung Software Engineering Lab, Noida, as requirements of semester project term for the award of degree of B.E. (Computer Science & Engineering), Thapar University, Patiala, under the guidance of Mr. Vijay Solanki and Prof. Anil Vashisht during 10<sup>th</sup> January, 2011 to 31<sup>st</sup> May, 2011.

Navjot Singh

Date: 31<sup>st</sup> May, 2011

10703066

Certified that the above statement made by the student is correct to the best of our knowledge and belief.

**Mr. Anil Vashisht**  
Asst. Professor  
Department Of Computer  
Science and Engineering

**Mr. Vijay Solanki**  
Chief Engineer  
Samsung India Software  
Engineering Lab, Noida

## **ACKNOWLEDGEMENT**

First and foremost I would like to thank Mr. Vijay Solanki, Software Engineering Lab Samsung Noida, who was also our supervisor, for his consistent guidance on each and every step of our project. Without his push and direction, this project would have not been completed. His continuous support and motivation made this project possible.

I am also highly grateful to Prof. Anil Vashisht for his excellent guidance and support and the entire faculty of the department of computer science and engineering, Thapar University, Patiala who were almost entirely responsible for providing the adequate base in subjects such as System Analysis and Design, Principles of Programming Languages etc. The project could never have been made without their kind support.

I'd also like to thank members of Android team Mr. Sachin Mittal, Mr. Sunil Rathour, Mr. Amitesh Abhay, Miss Swati Goel, Mr. Nitesh Goyal and Mr. Sandeep Bhadauria who helped me in understanding the basic concepts of android. They made sure that my understanding of Android is clear and deep.

## CONTENTS

COMPANY PROFILE .....	6
NOIDA LAB.....	7
JOURNEY .....	8
SAMSUNG GLOBAL.....	8
SAMSUNG ELECTRONICS HISTORY.....	10
SAMSUNG ELECTRONICS TODAY .....	10
MANAGEMENT PHILOSOPHY .....	11
SAMSUNG'S VISION 2020.....	12
SAMSUNG'S VALUES.....	12
TECHNOLOGY .....	13
Wireless.....	13
FUTURE TREND AND NEW TECHNOLOGIES .....	14
SOLUTIONS .....	15
MILESTONES AND AWARDS.....	15
ANDROID .....	16
FEATURES OF ANDROID.....	18
THE ANDROID PLATFORM.....	19
NATIVE ANDROID APPLICATIONS.....	21
ANDROID COMPONENTS .....	21
ANDROID SDK FEATURES .....	24
ANDROID SOFTWARE STACK .....	26
ANDROID APPLICATION.....	30
ACTIVITY.....	30
SERVICE.....	30
BROADCAST RECEIVER.....	30
CONTENT PROVIDERS.....	31
ACTIVITY LIFECYCLE OF ANDROID APPLICATION .....	31
LIFE CYCLE STATES .....	33
LIFE CYCLE EVENTS.....	35
APPLICATIONS DEVELOPED.....	39
STUDENT INFORMATION .....	42
SQLite in Android.....	42
ListView.....	45
APPLICATION FEATURES .....	47
Flow Diagram .....	47
SNAP-SHOTS .....	48
Work Program.....	57
Test cases .....	58
Conclusion and Future Scope .....	58
References .....	59
CONTACT TRACKER.....	60
ContactsContract API .....	60
ContactsContract.RawContacts .....	63
Class Overview .....	63

ContactsContract.Contacts.....	64
APPLICATION FEATURES .....	65
Flow Diagram .....	65
Work Program.....	77
Test Cases .....	78
Conclusion and Future Scope .....	78
References .....	78
DEMO APPLICATION FOR NOTIFICATION.....	79
Introduction to Notifications .....	79
THE APPLICATION.....	82
Flow Diagram .....	82
Snap-Shots .....	83
Work Program.....	87
References .....	87
DEMO APPLICATION ON GESTURES.....	88
References .....	89
RECENTAPPLICATIONS.....	90
Introduction.....	90
APPLICATION FEATURES .....	92
Work Program.....	92
References .....	93

## COMPANY PROFILE



The company's thrust on Product Innovation and R&D has given the company a competitive edge in the marketplace. Samsung has three Software development centres

- Samsung Electronics Lab (SEL) /Noida Lab,
- Samsung India Software Centre (SISC) ,Noida
- Samsung India Software operations unit (SISO) , Bangalore

Samsung India Software Centre is developing software solutions in Samsung's global software requirements for hi-end televisions like Plasma and LCD TVs and Digital Media Products, SISO is working on major projects for Samsung Electronics in the area of telecom: wireless terminals and infrastructure, Networking, SoC (System on Chip) Digital Printing and other multimedia/digital media as well as application software. In addition to working on global R&D projects, SISO is also helping Samsung India's Mobile business by focusing on product customisation for the Indian market. Samsung India currently employs around 2000 employees across its R&D Centres at Noida and Bangalore.

Samsung India is also carrying out Hardware R&D at its Noida R&D Centre. The focus of the R&D Centre is to customise both Consumer Electronics and Home Appliance products to better meet the needs of Indian consumers. From Flat televisions with 'Easy View' technology, Frost free refrigerators with Stabiliser free operations to Semi automatic washing machines with Silver Nano technology, the Samsung R&D Centres in India are helping the company to continuously innovate and introduce products customised for the Indian market

## World's largest conglomerate



### **NOIDA LAB**

Founded in 2007, Noida Lab was conceptualized with a vision to provide a competitive R&D platform in wireless technology. Headquartered in Noida, aligning with Samsung's vision, Noida Lab is one of the fastest-growing companies in the mobile software sector. Noida Lab produces efficient and dynamic mobile software that continually pushes the boundaries of innovation to provide true mobile freedom.

## **JOURNEY**

Samsung has established Software Engineering Lab in 2007 in India with a firm belief towards success and the realistic goals. Noida Lab has transformed itself into one the most competitive and proficient R&D centers of Samsung worldwide. It is focused on embedded and PC software development for Samsung Electronics, in a variety of areas related to Mobile Phones.

Noida Lab started its operations with Regional Adaptation and Multimedia testing while focusing to bring great innovations and latest technologies in the era of wireless communications. Noida Lab is engaged in development of latest mobile software for all range of handsets. Noida Lab also focuses on Quality assurance which includes automation Testing, protocol testing, white box testing and GCF certifications.

## **SAMSUNG GLOBAL**

Samsung has been moving continuously with a growing brand value in the world through advanced technologies like petrochemicals, fashion, medicines, semiconductors, skyscraper and plant construction, finance, hotels and many more for more than 70 years. Samsung is the place where innovation and quality is always the first priority. Samsung has achieved many milestones through a strong commitment towards quality, commitment for its values and vision and with the professional and empathetic approach with its partners, clients and its employees.

Globally, Samsung has been ranked among the top 30 largest employers in the world with more than 245000 employees. Samsung's scale is similar to the entire Singaporean economy with approximate revenue of \$174.2bn. Continuously standing as the largest driver of the Korean economy, Samsung holds 18% of Korean GDP and 20% of Korea export.

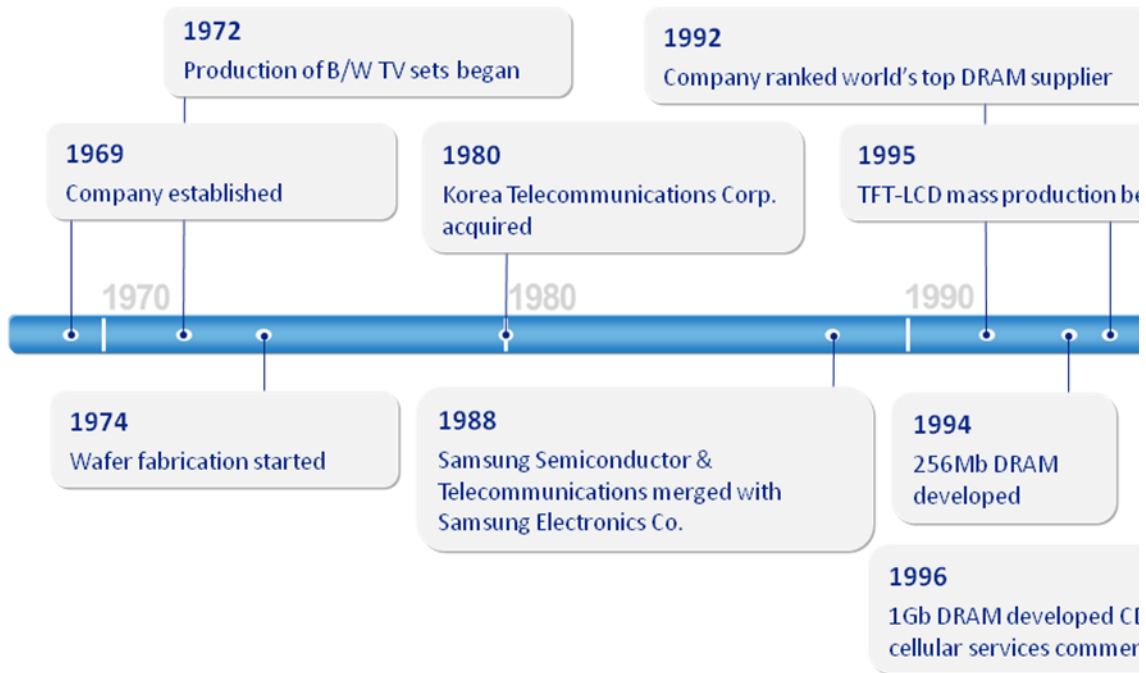


Former Chairman Mr. BC Lee, 50 years (1938-1987), founded Samsung, 1938 making it the Engine of Korean Economy. The Achiever Chairman Mr. KH Lee has been continuously making Samsung best in class Company since 1988 till today.

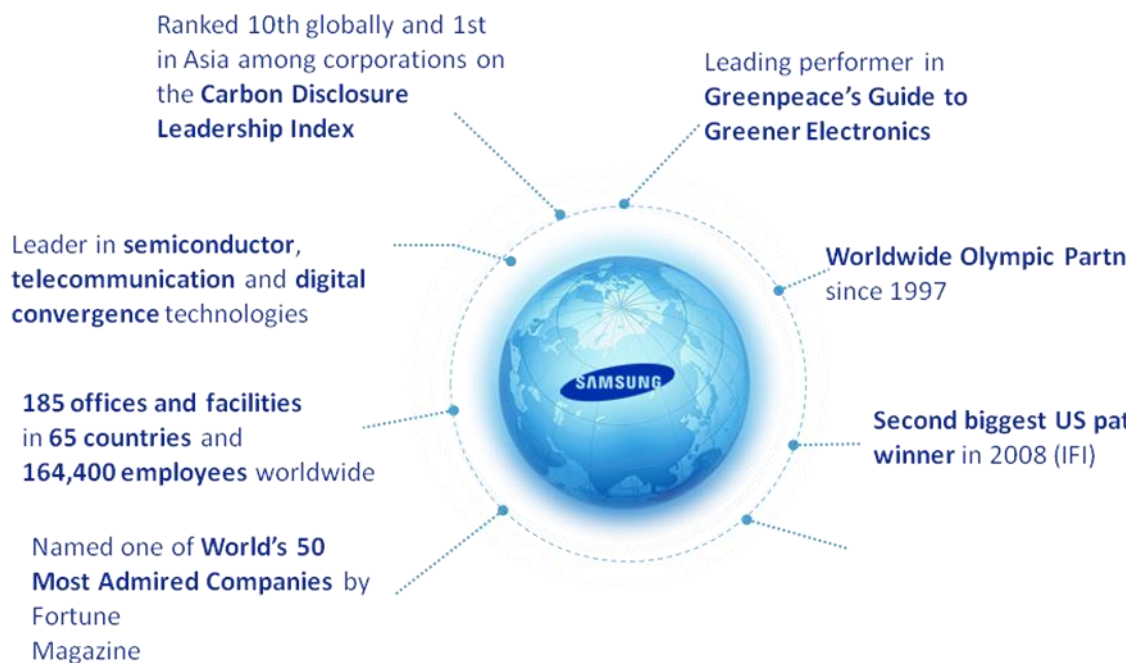
Samsung global has more than 429 offices/facilities 68 countries, including 30 Manufacturing Subsidiaries, 51 Sales Subsidiaries, one Manufacturing and Sales Subsidiaries, two Distribution Centers, six Design Centers, 18 R&D Centers, 77 Branch Offices and others.

With a strong commitment towards R&D, Samsung has invested \$5.3bn in 2005 gradually increasing to \$6.2bn in 2009.

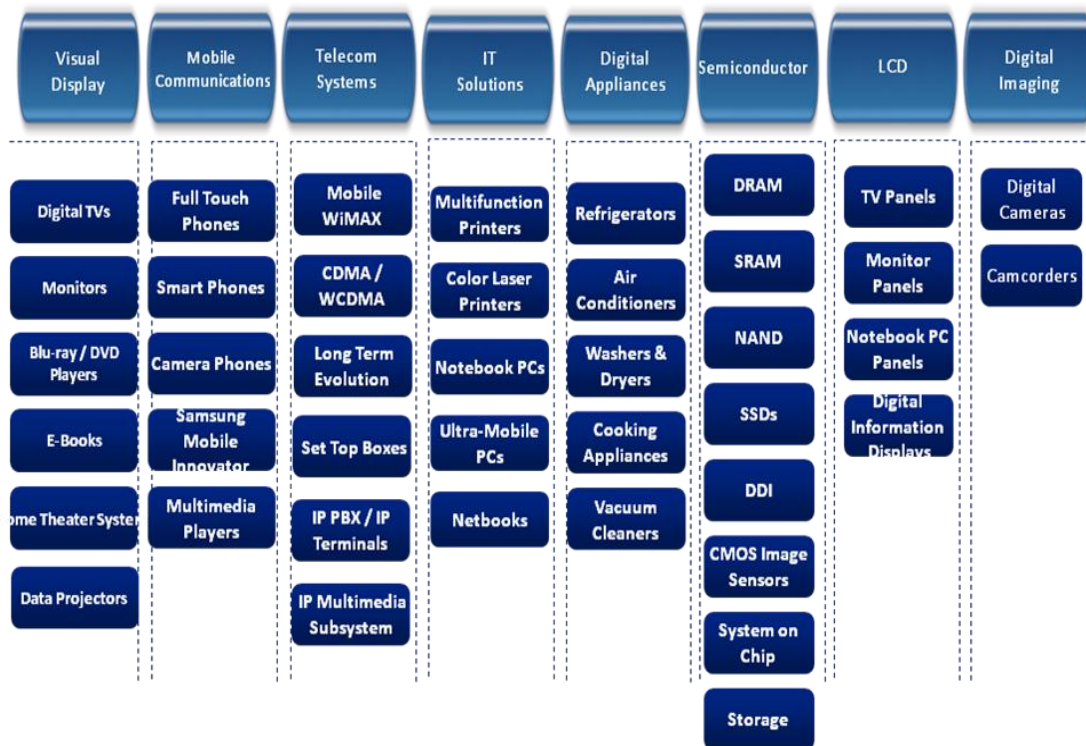
## SAMSUNG ELECTRONICS HISTORY



## SAMSUNG ELECTRONICS TODAY



## Samsung' Electronics: Businesses



| 10

## MANAGEMENT PHILOSOPHY

We at Noida Lab are focused to deliver high quality product using the combination of talent and technology. We believe that the utilization of transparent, collective and customer-focused approach of work can develop more value to our work.

In order to construct the organizational values and strengths, we at Noida Lab, believe in maintaining the great work place for experienced as well as new talents. We firmly believe in making our product, first choice for the customers.

## SAMSUNG'S VISION 2020

At Noida Lab, we aim to emerge as a centre of excellence for research, development, testing and solutions in wireless communication. Our target is to expand strong technological base in South West Asia region for carrying out research work, nurturing the local talent and exploring the market.

By following these efforts, Noida Lab hopes to contribute to a better world and a richer experience for all.

Vision of Noida Lab includes, following three strategies:

- **Technology:** The first and foremost objective of Noida Lab is to make optimum use of best technology for the research and development.
- **Innovation:** We believe in constant and continuous improvisation of techniques, methods and processes to obtain the innovative product.
- **Creative Solutions:** To make the product more customer-centric, we follow new ideas and provide creative solutions for additional features.

## SAMSUNG'S VALUES

We believe that living by strong values is the key to good business. At Samsung, a rigorous code of conduct and these core values are at the heart of every decision we make.

- **People:** Quite simply, a company is its people. At Samsung, we're dedicated to giving our people a wealth of opportunities to reach their full potential.
- **Excellence:** Everything we do at Samsung is driven by an unyielding passion for excellence—and an unfaltering commitment to develop the best products and services on the market.
- **Change:** In today's fast-paced global economy, change is constant and innovation is critical to a company's survival. We set our sights on the future,

anticipating market needs and demands so we can steer our company toward long-term success.

- **Integrity:** Operating in an ethical way is the foundation of our business. Everything we do is guided by a moral compass that ensures fairness, respect for all stakeholders and complete transparency.
- **Co-prosperity:** A business cannot be successful unless it creates prosperity and opportunity for others. Samsung is dedicated to being a socially and environmentally responsible corporate citizen in every community where we operate around the globe.

## TECHNOLOGY

### Wireless

Noida Lab is involved in development and testing of extensive range of mobiles software by the combination of exceptional talent and superior technology.

The focus of Noida Lab center is to produce best quality mobile phone software in both CDMA and GSM technologies. The center has talented engineers for carrying out R&D works having development and quality teams. The research and development activities are carried out in ideal manner to stabilize every model in terms of quality, features and enhancement. The dedicated team works in coordination with other centers across the world to produce high quality mobiles considering various factors such as market research, target customers, competing products etc.

## **KEY TECHNOLOGIES FEATURE OF NOIDA LAB R&D**

- Mobile platforms (BADA, SHP, LIMO, MMP, NXP, SAP (Samsung AJU platform), Android) and device driver.
- MMI customization
- Mobile features and application development
- Java TCK and J2ME application development
- Tools development for Mobile application development
- Protocols and standards such as E-mail, MMS, Sync ML, XML, GCF etc.

## **FUTURE TREND AND NEW TECHNOLOGIES**

Noida Lab is continuously strengthening its skills sets on the future technologies such as 3G and 4G. The centre is constantly engaged in bada application development which is going to be the platform for the future smart phones, containing hundreds of applications. These versatile phones have proved a success in the world market with affordable price, magnificent features and diverse variety applications.

## **SOLUTIONS**

Apart from research and development, Noida Lab is also occupied in the creations of new ideas and providing solutions by analyzing the customer's view point and prospective. For India region, the centre is involved in following solutions:

- Mobile Prayer
- Indian Calendar
- Mobile Tracker
- Dictionary

## **MILESTONES AND AWARDS**

- Brand Value Ranked 19th in the World in 2009.
- Selected as the fastest growing "Winner Brand" for two consecutive years (2002-2003).
- It has been the fastest growing global brand since 2002.
- Samsung is the 2nd Top Patent Winner (2009, US), holding more than 3611 patents till date.
- Launched World's Slimmest LED TV (Jan, 2009).
- World's First HSUPA Phone (Apr, 2008).
- World's First 30nm 64 GB NAND (2007).
- World's First Blu-ray Player (Jun, 2006).

- World's First HSPDA Phone (May, 2006).
- World's Largest TV (Sep, 2005).
- Established Noida Lab in 2007.

## **ANDROID**

Android is a software platform from Google and the Open Handset Alliance that has the potential to revolutionize the global cell phone market. It has the capability to make inroads in many other (non-phone) embedded application in markets.

Android consists of complete set of software components for mobile devices including:

- an operating system,
- middleware, and
- Key mobile applications.

Whether you're an experienced mobile engineer, a desktop or web developer, or a complete programming novice, Android represents an exciting new opportunity to write innovative applications for mobile devices.

Despite the name, Android will not help you create an unstoppable army of emotionless robot warriors on a relentless quest to cleanse the earth of the scourge of humanity.

Instead, *Android* is an open source software stack that includes the operating system, middleware, and key applications along with a set of API libraries for writing mobile applications that can shape the look, feel, and function of mobile handsets. Small, stylish, and versatile, modern mobile phones have become powerful tools that incorporate cameras, media players, GPS systems, and touch screens. As technology has evolved, mobile devices have become about more than simply making calls, but their software and development platforms have struggled to keep pace.

Until recently, mobile phones were largely closed environments built on proprietary operating systems that required proprietary development tools. The phones themselves



often prioritized native applications over those written by third parties. This has introduced an artificial barrier for developers hoping to build on increasingly powerful mobile hardware. In Android, native and third-party applications are written using the same APIs and executed on the same run time. These APIs feature hardware access, location-based services, support for background services, map-based activities, relational databases, interdevice peer-to-peer messaging, and 2D and 3D graphics. Android has powerful APIs, excellent documentation, a thriving developer community, and no development or distribution costs. As mobile devices continue to increase in popularity, this is an exciting opportunity to create innovative mobile phone applications no matter what your development background.

### **What is Open Handset Alliance (OHA)?**

The Open Handset Alliance is a business alliance of 80 firms to develop open standards for mobile devices.

These firms together have developed Android, the first complete, open and free mobile platform.

Member firms include Google, Sony, Dell, Intel, Motorola, Samsung and many more.

OHA represents

“A commitment to openness, a shared vision for the future, and concrete plans to make vision a reality. To accelerate innovation in mobile and offers consumers a richer, less expensive, and better mobile experience.”

The OHA hopes to deliver a better mobile software experience for consumers by providing the platform needed for innovative mobile development at a faster rate and a higher quality without licensing fees for software developers or handset manufacturers.

Ultimately the success of Android as a mobile platform will depend largely on the success of OHA partners in releasing desirable handsets and mobile services that encourage the widespread adoption of Android phones. Developers meanwhile have the opportunity to create innovative new mobile applications for Android to encourage more mobile technology companies to become part of the OHA.

## **FEATURES OF ANDROID**

### **1. Open**

Android was built from ground-up to enable developers to create compelling mobile application that take full advantage all a handset has to offer. It was built to be truly open. For example, an application can call upon any of the phones core functionality such as making calls, sending text messages, or using the camera, allowing user to create richer and more cohesive experiences of 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.

### **2. 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 home screen, 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.

### **3. 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.

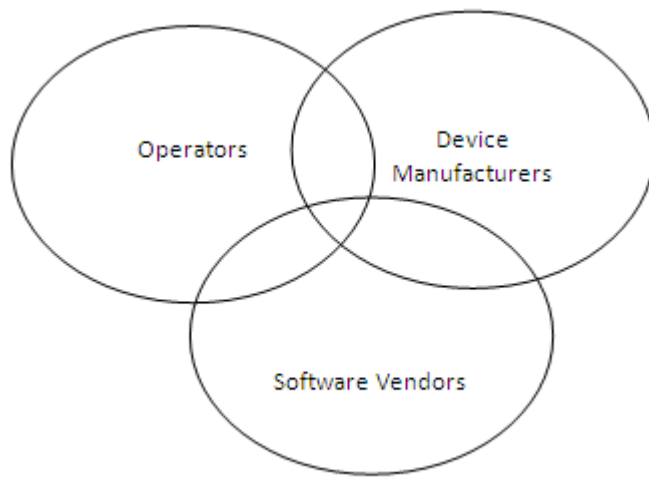
#### **4. Fast and easy development**

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.

#### **THE ANDROID PLATFORM**

- Android is software environment built for mobile devices.
- It is not hardware platform.
- Android includes:
  - Linux kernel-based OS,
  - A rich UI,
  - Telephone functionality,
  - End-user applications,
  - Code libraries,
  - Application frameworks,
  - Multimedia support ...
- User applications are built for android on Java.

## Android's Context: Mobile Market Players



Mobile **network operators** want to lock down their networks, controlling and metering traffic.

**Device manufacturers** want to differentiate themselves with features, reliability, and price points.

**Software vendors** want complete access to the hardware to deliver cutting edge applications.

## Android vs. Competitors

1. Apple's IOS
2. Microsoft's Windows for Phone
3. Nokia's Symbian
4. Palm OS
5. Blackberry's Research In Motion

## NATIVE ANDROID APPLICATIONS

Android phones will normally come with a suite of preinstalled applications including, but not limited to:

- An e-mail client compatible with Gmail but not limited to it
- An SMS management application
- A full PIM (personal information management) suite including a calendar and contacts list, both tightly integrated with Google's online services
- A fully featured mobile Google Maps application including Street View, business finder, driving directions, satellite view, and traffic conditions
- A Web Kit-based web browser
- An Instant Messaging Client
- A music player and picture viewer
- The Android Marketplace client for downloading third-party Android applications.
- The Amazon MP3 store client for purchasing DRM free music.

All the native applications are written in Java using the Android SDK and are run on Dalvik Virtual Machine (DVM).

The data stored and used by the native applications — like contact details — are also available to third-party applications. Similarly, your applications can handle events such as an incoming call or a new SMS message.

The exact makeup of the applications available on new Android phones is likely to vary based on the hardware manufacturer and/or the phone carrier or distributor.

## ANDROID COMPONENTS

- **Application framework** enabling reuse and replacement of components
- **Dalvik virtual machine** optimized for mobile devices
- **Integrated browser** based on the open source **Web Kit** engine
- **Optimized graphics** powered by a custom 2D graphics library; 3D graphics

based on the OpenGL ES 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 Wi-Fi** (hardware dependent)
- **Camera, GPS, compass, and accelerometer** (hardware dependent)

### **What Does It Have That Others Don't?**

Many of the features listed previously, such as 3D graphics and native database support, are also available

in other mobile SDKs. Here are some of the unique features that set Android apart:

❑ **Google Map Applications** Google Maps for Mobile has been hugely popular, and android offers a Google Map as an atomic, reusable control for use in your applications. The Map View widget lets you display, manipulate, and annotate a Google Map within your Activities to build map-based applications using the familiar Google Maps interface.

❑ **Background Services and Applications** Background services let you create applications that use an event-driven model, working silently while other applications are being used or while your mobile sits ignored until it rings, flashes, or vibrates to get your attention. Maybe it's an application that tracks the stock market, alerting you to significant changes in your portfolio, or a service that changes your ring tone or volume depending on your current location, the time of day, and the identity of the caller.

❑ **Shared Data and Interprocess Communication** Using Intents and Content Providers, Android lets your applications exchange messages, perform processing, and share data. You can also use these mechanisms to leverage the data and functionality provided by the native Android applications. To mitigate the risks of such an open strategy, each application's process, data storage, and files are private unless explicitly shared with other applications using a full permission-based security mechanism.

- ❑ **All Applications Are Created Equal** Android doesn't differentiate between native applications and those developed by third parties. This gives consumers unprecedented power to change the look and feel of their devices by letting them completely replace every native application with a third-party alternative that has access to the same underlying data and hardware. Every rule needs an exception and this one has two. The “unlock” and “in-call experience” screens cannot be replaced in the initial SDK release.
- ❑ **P2P Inter-device Application Messaging** Android offers peer-to-peer messaging that supports presence, instant messaging, and inter-device/inter-application communication.

### **Introducing Development Framework**

Android applications are written using Java as a programming language but are executed using a custom virtual machine called *Dalvik* rather than a traditional Java VM. Each Android application runs in a separate process within its own Dalvik instance, relinquishing all responsibility for memory and process management to the Android run time, which stops and kills processes as necessary to manage resources.

Dalvik and the Android run time sit on top of a Linux kernel that handles low-level hardware interaction including drivers and memory management, while a set of APIs provides access to all of the underlying services, features, and hardware.

### **What Comes with SDK**

The Android software development kit (SDK) includes everything you need to start developing, testing, and debugging Android applications.

Included in the SDK download are:

- ❑ **The Android APIs** The core of the SDK is the Android API libraries that provide developer access to the Android stack. These are the same libraries used at Google to create native Android applications.
- ❑ **Development Tools** To turn Android source code into executable Android applications, the SDK includes several development tools that let you compile and debug your applications.
- ❑ **The Android Emulator** The Android Emulator is a fully interactive Android device

emulator featuring several alternative skins. Using the emulator, you can see how your applications will look and behave on a real Android device. All Android applications run within the Dalvik VM so that the software emulator is an excellent environment — in fact, as it is hardware-neutral, it provides a better independent test environment than any single hardware implementation.

❑ **Full Documentation** The SDK includes extensive code-level reference information detailing exactly what's included in each package and class and how to use them. In addition to the code documentation, Android's reference documentation explains how to get started and gives detailed explanations of the fundamentals behind Android development.

❑ **Sample Code** The Android SDK includes a selection of sample applications that demonstrate some of the possibilities available using Android, as well as simple programs that highlight how to use individual API features.

❑ **Online Support** Despite its relative youth, Android has generated a vibrant developer community. The Google Groups at <http://code.google.com/android/groups> are active forums of Android developers with regular input from the Android development team at Google.

## ANDROID SDK FEATURES

The true appeal of Android as a development environment lies in the APIs it provides. As an application-neutral platform, Android gives you the opportunity to create applications that are as much a part of the phone as anything provided out of the box.

The following list highlights some of the most noteworthy Android features:

- No licensing, distribution, or development fees
- Wi-Fi hardware access
- GSM, EDGE, and 3G networks for telephony or data transfer, allowing you to make or receive calls or SMS messages, or to send and retrieve data across mobile networks
- Comprehensive APIs for location-based services such as GPS
- Full multimedia hardware control including playback and recording using the



camera and microphone

- APIs for accelerometer and compass hardware
- IPC message passing
- Shared data stores
- An integrated open source Web Kit-based browser
- Full support for applications that integrate Map controls as part of their user interface
- Peer-to-peer (P2P) support using Google Talk
- Mobile-optimized hardware-accelerated graphics including a path-based 2D graphics library and support for 3D graphics using OpenGL ES
- Media libraries for playing and recording a variety of audio/video or still image formats
- An application framework that encourages reuse of application components and the replacement of native applications.

## ANDROID SOFTWARE STACK

The Android software stack is composed of the elements shown in Figure and described in further detail below it. Put simply, a Linux kernel and a collection of C/C++ libraries are exposed through an application framework that provides services for, and management of, the run time and applications.



❑ **Linux Kernel** Core services (including hardware drivers, process and memory management, security, network, and power management) are handled by a Linux 2.6 kernel. The kernel also provides an abstraction layer between the hardware and the remainder of the stack.

❑ **Libraries** Running on top of the kernel, Android includes various C/C++ core libraries such as libc and SSL, as well as:

- ❑ A media library for playback of audio and video media

- ❑ A Surface manager to provide display management
  - ❑ Graphics libraries that include SGL and OpenGL for 2D and 3D graphics
  - ❑ SQLite for native database support
  - ❑ SSL and WebKit for integrated web browser and Internet security
- ❑ **Android Run Time** what makes an Android phone an Android phone rather than a mobile Linux implementation is the Android run time. Including the core libraries and the Dalvik virtual machine, the Android run time is the engine that powers your applications and, along with the libraries, forms the basis for the application framework.
- ❑ **Core Libraries** While Android development is done in Java, Dalvik is not a Java VM. The core Android libraries provide most of the functionality available in the core Java libraries as well as the Android-specific libraries.
  - ❑ **Dalvik Virtual Machine** Dalvik is a register-based virtual machine that's been optimized to ensure that a device can run multiple instances efficiently. It relies on the Linux kernel for threading and low-level memory management.
  - ❑ **Application Framework** The application framework provides the classes used to create Android applications. It also provides a generic abstraction for hardware access and manages the user interface and application resources.
  - ❑ **Application Layer** All applications, both native and third party, are built on the application layer using the same API libraries. The application layer runs within the Android run time using the classes and services made available from the application framework.

## Android Libraries

The core libraries provide all the functionality you need to start creating applications for Android, but it won't be long before you're ready to delve into the advanced APIs that offer the really exciting functionality.

Android hopes to target a wide range of mobile hardware, so be aware that the suitability and implementation of the following APIs will vary depending on the device upon which they are implemented.

- ❑ **android.util** The core utility package contains low-level classes like specialized containers, string formatters, and XML parsing utilities.
- ❑ **android.os** The operating system package provides access to basic operating system services like message passing, interprocess communication, clock functions, and debugging.
- ❑ **android.graphics** The graphics API supplies the low-level graphics classes that support canvases, colors, and drawing primitives, and lets you draw on canvases.
- ❑ **android.text** The text processing tools for displaying and parsing text.
- ❑ **android.database** Supplies the low-level classes required for handling cursors when working with databases.
- ❑ **android.content** The content API is used to manage data access and publishing by providing services for dealing with resources, content providers, and packages.
- ❑ **android.view** Views are the core user interface class. All user interface elements are constructed using a series of Views to provide the user interaction components.
- ❑ **android.widget** Built on the View package, the widget classes are the “here’s one we created earlier” user-interface elements for you to use in your applications. They include lists, buttons, and layouts.
- ❑ **com.google.android.maps** A high-level API that provides access to native map controls that you can use within your application. Includes the MapView control as well as the Overlay and MapController classes used to annotate and control your embedded maps.
- ❑ **android.app** A high-level package that provides access to the application model. The application package includes the Activity and Service APIs that form the basis for all your Android applications.
- ❑ **android.provider** To ease developer access to certain standard Content Providers (such as the contacts database), the Provider package offers classes to provide access to standard databases included in all Android distributions.
- ❑ **android.telephony** The telephony APIs give you the ability to directly interact with the device’s phone stack, letting you make, receive, and monitor phone calls, phone

status, and SMS messages.

❑ **android.webkit** The WebKit package features APIs for working with Web-based content, including a WebView control for embedding browsers in your activities and a cookie manager

❑ **android.location** The location-based services API gives your applications access to the device's current physical location. Location-based services provide generic access to location information using whatever position-finding hardware or technology is available on the device.

❑ **android.media** The media APIs provide support for playback and recording of audio and video media files, including streamed media.

❑ **android.opengl** Android offers a powerful 3D rendering engine using the OpenGL ES API that you can use to create dynamic 3D user interfaces for your applications.

❑ **android.hardware** Where available, the hardware API exposes sensor hardware including the camera, accelerometer, and compass sensors.

❑ **android.bluetooth, android.net.wifi, and android.telephony** Android also provides low-level access to the hardware platform, including Bluetooth, Wi-Fi, and telephony hardware.

## ANDROID APPLICATION

An application consists of one or more *components* that are defined in the application's manifest file. A component can be one of the following:

1. *An Activity*
2. *A Service*
3. *A broadcast receiver*
4. *A content provider*

### ACTIVITY

An *activity* usually presents a *single visual user interface* from which a number of actions could be performed. Although activities work together to form a cohesive user interface, *each activity is independent of the others*. 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 through so called *intents*.

### SERVICE

*A service doesn't have a visual user interface, but rather runs in the background for an indefinite period of time*. It's possible to connect to (*bind to*) an ongoing service (and start the service if it's not already running). While connected, you can communicate with the service through an interface that the service exposes.

### BROADCAST RECEIVER

*A broadcast receiver* is a component that does nothing but *receives and reacts to broadcast announcements*. Many broadcasts originate in system code (e.g. "you got mail") but any other applications can also initiate broadcasts. *Broadcast receivers do not display a user interface*. However, they may start an activity in response to the information they receive, or - as services do - they may use the notification manager to alert the user.

## CONTENT PROVIDERS

A *content provider* makes a specific set of the application's data available to other applications. The data usually is stored in the file system, or in an SQLite database. The content provider implements 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 *content resolver* object and call its methods instead. A content resolver can talk to any content provider; it cooperates with the provider to manage any interprocess communication that's involved.

## ACTIVITY LIFECYCLE OF ANDROID APPLICATION

A Linux process encapsulating an Android application is created for the application when some of its code needs to be run, and will remain running until

1. It is no longer needed, OR
2. The system needs to reclaim its memory for use by other applications.

An unusual and fundamental feature of Android is that an application process's lifetime is not directly controlled by the application itself.

Instead, it is determined by the system through a combination of

1. The parts of the application that the system knows are running,
2. How important these things are to the user, and
3. How much overall memory is available in the system?

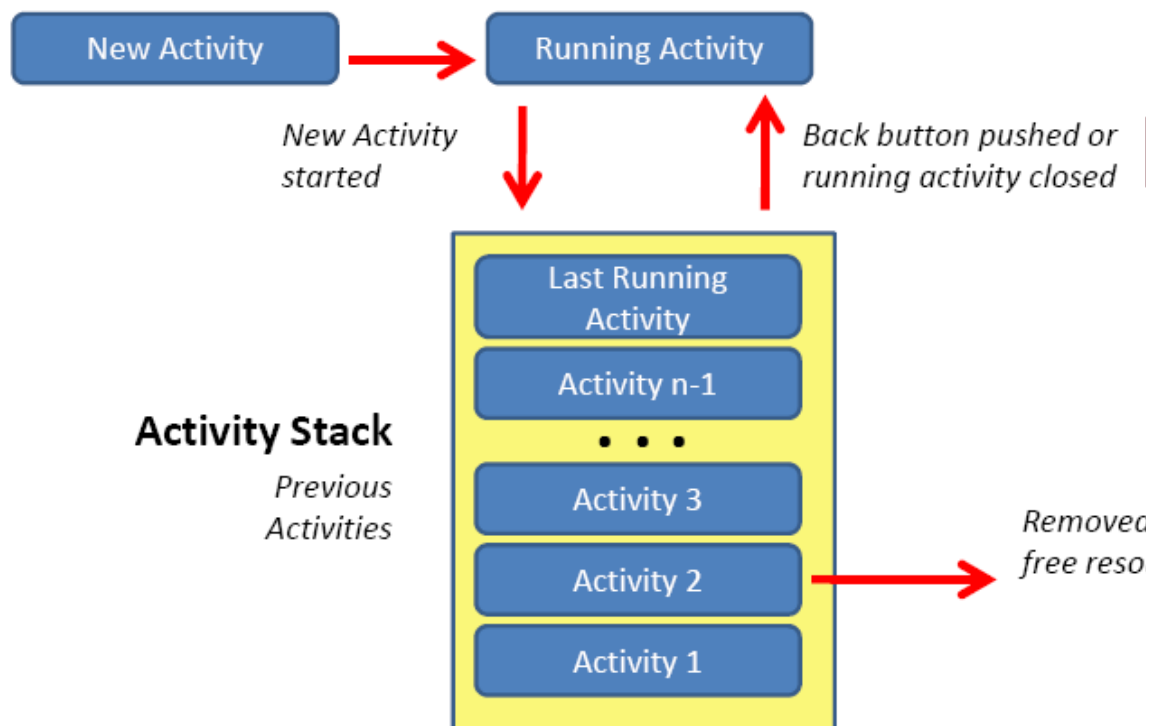
### Component Lifecycles

Application components have a lifecycle

1. A beginning when Android instantiates them to respond to intents through to an end when the instances are destroyed.
2. In between, they may sometimes be *active* or *inactive*, or - in the case of activities- *visible* to the user or *invisible*.

## Activity Stack

- Activities in the system are managed as an *activity stack*.
- When a new activity is *started*, it is placed on the *top* of the stack and becomes the running activity the previous activity always remains below it in the stack, and will not come to the foreground again until the new activity exits.
- If the user presses the *Back Button* the next activity on the stack moves up and becomes active.





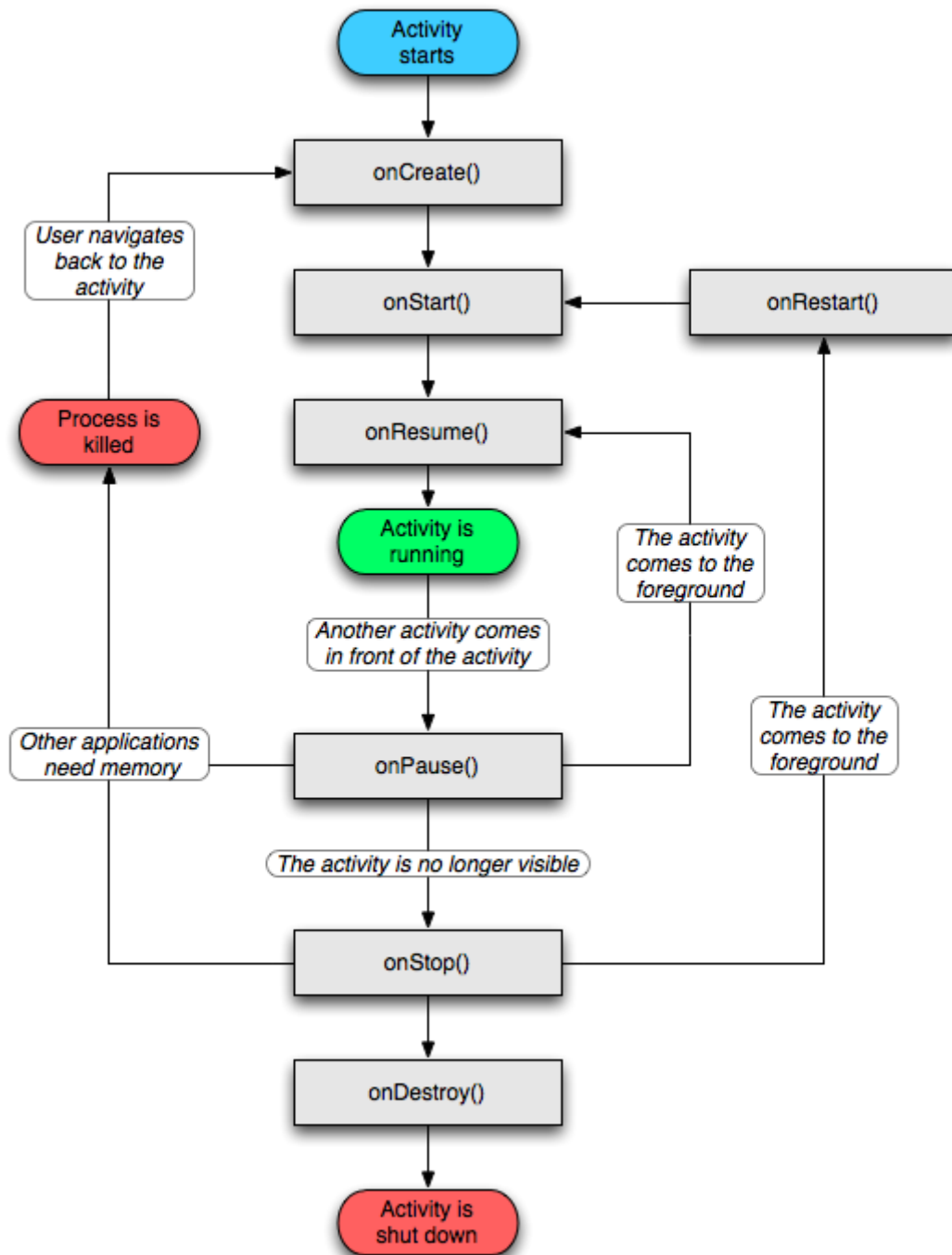
## LIFE CYCLE STATES

An activity has essentially three states:

1. It is *active* or *running*
2. It is *paused* or
3. It is *stopped*.

An activity has essentially three states:

1. It is ***active or running*** when it is in the *foreground* of the screen (at the top of the *activity stack* for the current task). This is the activity that is the focus for the user's actions.
2. It is ***paused*** if it has lost focus but is still visible to the user. That is, another activity lies on top of it and that new activity either is *transparent* or *doesn't cover the full screen*. A paused activity is completely *alive* (it maintains all state and member information and remains attached to the window manager), but can be killed by the system in extreme low memory situations.
3. It is ***stopped*** if it is completely *obscured* by another activity. It still retains all state and member information. However, *it is no longer visible* to the user so its window is hidden and it will often be killed by the system when memory is needed elsewhere.



## LIFE CYCLE EVENTS

If an activity is paused or stopped, the system can drop it from memory either by asking it to finish (calling its `finish()` method), or simply killing its process. When it is displayed again to the user, it must be completely restarted and restored to its previous state. As an activity transitions from state to state, it is notified of the change by calls to the following protected *transition* methods:

<code>void onCreate(Bundle savedInstanceState)</code> <code>void onStart()</code> <code>void onRestart()</code> <code>void onResume()</code>	<code>void onPause()</code> <code>void onStop()</code> <code>void onDestroy()</code>
---	--

All of these methods are hooks that you can override to do appropriate work when the state changes. All activities must implement `onCreate()` to do the initial setup when the object is first instantiated. Many activities will also implement `onPause()` to commit data changes and otherwise prepare to stop interacting with the user. The seven transition methods define the entire lifecycle of an activity. The entire lifetime of an activity happens between the first call to `onCreate()` through to a single final call to `onDestroy()`. An activity does all its initial setup of "global" state in `onCreate()`, and releases all remaining resources in `onDestroy()`. The visible lifetime of an activity happens between a call to `onStart()` until a corresponding call to `onStop()`. During this time, the user can see the activity on- screen, though it may not be in the foreground and interacting with the user. The `onStart()` and `onStop()` methods can be called multiple times, as the activity alternates between being visible and hidden to the user. Between these two methods, you can maintain resources that are needed to show the activity to the user.

## APPLICATION'S LIFE CYCLE FOREGROUND LIFETIME

The foreground lifetime of an activity happens between a call to `onResume()` until a corresponding call to `onPause()`. During this time, the activity is in front of all other activities on screen and is interacting with the user. An activity can frequently transition between the *resumed* and *paused* states — for example,

- `onPause()` is called when the device goes to sleep or when a new activity is started,
- `onResume()` is called when an activity result or a new intent is delivered. The

foreground lifetime of an activity happens between a call to `onResume()` until a corresponding call to `onPause()`. During this time, the activity is in front of all other activities on screen and is interacting with the user. An activity can frequently transition between the *resumed* and *paused* states — for example,

### **Method: onCreate()**

- Called when the activity is first created.
- This is where you should do all of your normal static set up create views, bind data to lists, and so on.
- This method is passed a *Bundle* object containing the activity's previous state, if that state was captured.
- Always followed by `onStart()`

### **Method: onRestart()**

- Called after the activity has been stopped, just prior to it being started again.
- Always followed by `onStart()`

### **Method: onStart()**

- Called just before the activity becomes visible to the user.
- Followed by `onResume()` if the activity comes to the foreground, or `onStop()` if it becomes hidden.

**Method: onResume()**

1. Called just before the activity starts interacting with the user.
2. At this point the activity is at the top of the activity stack, with user input going to it.
3. Always followed by *onPause()*.

**Method: onPause()**

1. Called when the system is about to start resuming another activity.
2. This method is typically used to *commit* unsaved changes to persistent data, stop animations and other things that may be consuming CPU, and so on.
3. It should do whatever it does very quickly, because the next activity will not be resumed until it returns.
4. Followed either by *onResume()* if the activity returns back to the front, or by *onStop()* if it becomes invisible to the user.
5. The activity in this state is *killable* by the system.

**Method: onStop()**

1. Called when the activity is no longer visible to the user.
2. This may happen because it is being destroyed, or because another activity (either an existing one or a new one) has been resumed and is covering it.
3. Followed either by *onRestart()* if the activity is coming back to interact with the user, or by *onDestroy()* if this activity is going away.
4. The activity in this state is *killable* by the system.

**Method: onDestroy()**

1. Called before the activity is destroyed.
2. This is the final call that the activity will receive.
3. It could be called either because the activity is finishing (someone called *finish()* on it), or because the system is temporarily destroying this instance of the activity to save space.
4. You can distinguish between these two scenarios with the *isFinishing()* method.
5. The activity in this state is *killable* by the system.

## Killable States

- Activities on killable states can be terminated by the system *at any time after the method returns, without executing another line of the activity's code.*
- Three methods (*onPause()*, *onStop()*, and *onDestroy()*) are *killable*.
- *onPause()* is the only one that is guaranteed to be called before the process is killed — *onStop()* and *onDestroy()* may not be.
- Therefore, you should use *onPause()* to write any persistent data (such as user edits) to storage.

## **APPLICATIONS DEVELOPED**

- Student Information
- Contact Tracker
- Demo Application for Notifications
- Demo Application for Gestures
- “Recent Applications” for T Mobile’s SideKick Device.

## **Requirements**

The system and software requirements for developing Android applications using the Android SDK.

## **Supported 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 Ubuntu Linux, Lucid Lynx)
  - GNU C Library (glibc) 2.7 or later is required.
  - On Ubuntu Linux, version 8.04 or later is required.
  - 64-bit distributions must be capable of running 32-bit applications.

## **Supported Development Environments**

### **Eclipse IDE**

- Eclipse 3.4 (Ganymede) or greater
- Eclipse JDT 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 Developers
- Eclipse Classic (versions 3.5.1 and higher)
- Eclipse IDE for Java EE Developers
- JDK 5 or JDK 6 (JRE alone is not sufficient)
- Android Development Tools plugin (recommended)
- **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 1.8 or later
- **Not** compatible with Gnu Compiler for Java (gcj)

### **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	35 MB	Required.
SDK Platform-tools	6 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.

## STUDENT INFORMATION

Student Information system is an Android application which lets you add, delete and search, a student's information into a SQLite database. The information the following fields:

- First Name
- Middle Name
- Last Name
- Degree
- Specialization
- Father Name
- Mother Name
- Date of Birth
- Email Id
- Contact Number

### SQLite in Android

[SQLite](#) is an Open Source Database which is embedded into Android. SQLite supports standard relational database features like [SQL](#) syntax, transactions and prepared statements. In addition it requires only little memory at runtime (approx. 250 KByte).

Using SQLite in Android does not require any database setup or administration. You specify the SQL for working with the database and the database is automatically managed for you.

SQLite supports the data types TEXT (similar to String in Java), INTEGER (similar to long in Java) and REAL (similar to double in Java). All other types must be converted into one of these fields before saving them in the database. SQLite itself does not validate

if the types written to the columns are actually of the defined type, you can write an integer into a string column.

If your application creates a database this database is saved in the directory "DATA/data/APP\_NAME/databases/FILENAME". "DATA" is the path which Environment.getDataDirectory() returns, "APP\_NAME" is your application name and "FILENAME" is the name you give the database during creation. Environment.getDataDirectory() usually return the SD card as location.

### ➤ **SQLiteOpenHelper**

To create and upgrade a database in your Android application you usually subclass "SQLiteOpenHelper". In this class you need to override the methods onCreate() to create the database and onUpgrade to upgrade the database in case of changes in the database schema. Both methods receive an "SQLiteDatabase" object.

"SQLiteDatabase" provides the execSQL() method which allows to execute SQL.

It also provides the methods getReadableDatabase() and getWritableDatabase() to get access to the database either in read or write mode. The database is represented by an "SQLiteDatabase" object.

For the primary key of the database you should always use the identifier "\_id" as some of Android functions rely on this standard.

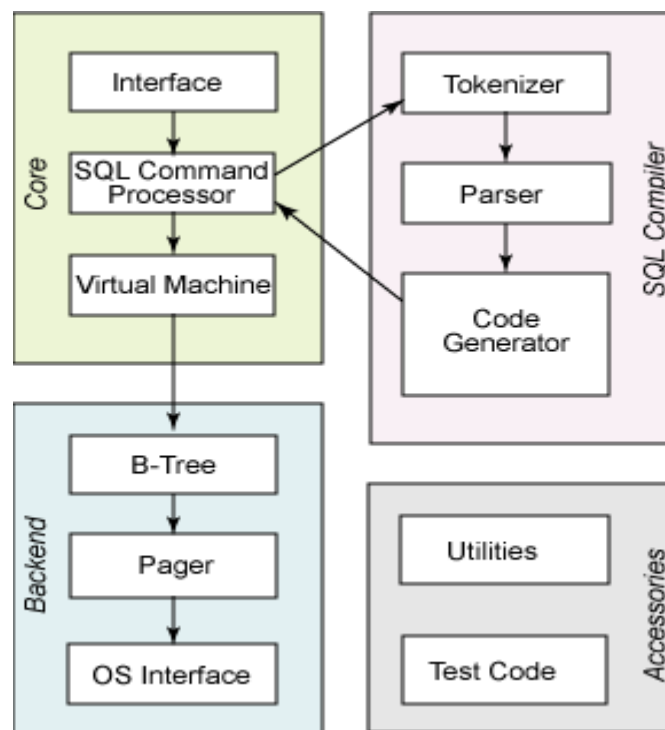
### ➤ **SQLiteDatabase and Cursor**

"SQLiteDatabase" provides the methods insert(), update() and delete().

Queries can be created via the method rawQuery() which accepts SQL, query() which provides an interface for specifying dynamic data or SQLiteQueryBuilder. SQLiteBuilder is similar to the interface of an content provider therefore it is typically used for them. A query returns always a "Cursor".

A Cursor represents the result of a query. To get the number of elements use the method `getCount()`. To move between individual data rows, you can use the methods `moveToFirst()` and `moveToNext()`. Via the method `isAfterLast()` you can check if there is still some data.

A Cursor can be directly used via the "SimpleCursorAdapter" in ListViews.



Internal architecture of SQLite

## **ListView**

[Android](#) provides the [view](#) "ListView" which is capable of displaying a scrollable list of items. "ListView" gets the data to display via an adapter. An adapter which must extend "BaseAdapter" and is responsible for providing the data model for the list and for converting the data into the fields of the list.

Android has two standard adapters, ArrayAdapter and [CursorAdapter](#) . "ArrayAdapter" can handle data based on Arrays or Lists while "SimpleCursorAdapter" handle [database](#) related data. You can develop your own Adapter by extending these classes or the BaseAdapter class.

### **➤ ListActivity**

You can directly use the "ListView" in your layout as any other UI component. In case your [Activity](#) is primary showing a list you can extend the activity "ListActivity" which simplifies the handling of a "ListView". "ListActivity" extends "Activity" and provides simplified handling of lists. For example you have a predefined method if someone clicks on a list element.

"ListActivity" contains a "ListAdapter" which is responsible for managing the data. This adapter must be set in the onCreate() method of your Activity via the method setListAdapter().

If the user select in the list a list entry the method onItemClick() will be called. This method allows to access the selected element.

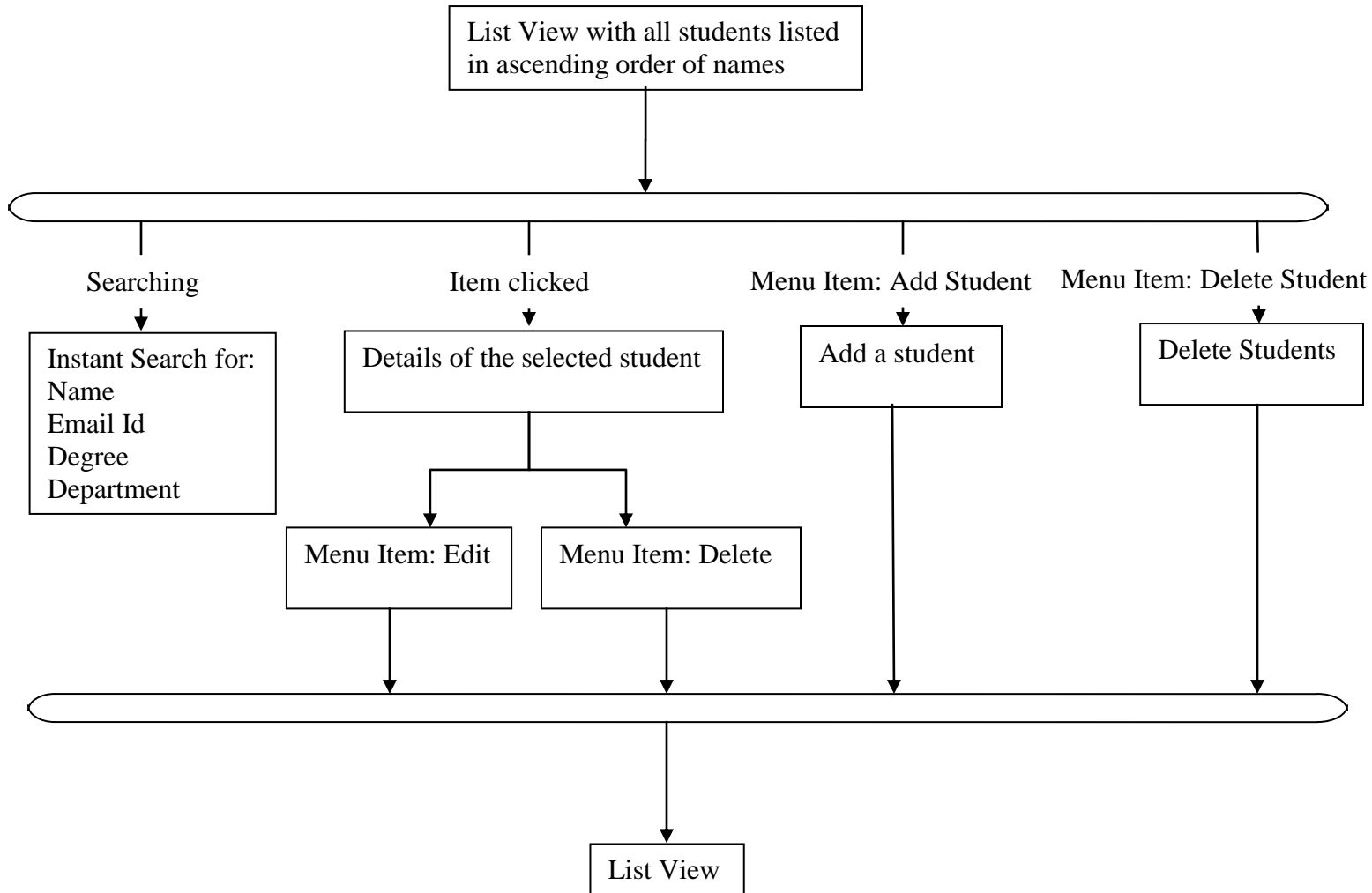
Android provides already some default layouts which you can use in your Adapter, e.g. "android.R.layout.simple\_list\_item1". In case you don't want to use one of the predefined layouts your own layout must have an element with the id "@android:id/list" which is the ListView. You can also use a view with the id "@android:id/empty". This view is displayed if the list is empty. For example you could display here an error message.

### ➤ **ListViews and performance**


Displaying a large dataset must be efficiently implemented on a mobile device. Therefore the ListView only creates views (widget) if needed and attach them to the view hierarchy. The default Adapter implementation for a ListView will recycle views, e.g. if a row is not displayed anymore it will be recycled and only its content will change. If you implement your own adapter for a view you also should do this to avoid performance problems.

## APPLICATION FEATURES

### Flow Diagram



## SNAP-SHOTS



The screenshot shows a mobile application interface with a dark background and a green digital rain effect. At the top, there is a status bar with icons for email, USB, Android, and battery, along with the time 1:14 AM. Below the status bar is a header bar with the title 'STUDENT INFO.'. Under the header, there are three tabs: 'By name or email', 'By degree', and 'By specialization'. The 'By name or email' tab is selected, and it contains a search bar with the text 'Email Id' and a dropdown arrow. The 'By degree' and 'By specialization' tabs also have dropdown arrows. Below the search bar is a list of student records, each consisting of a name and an email address, separated by a horizontal line. The records are: Akash Goel (akash.g@samsung.com), Maninder Arora (maninder.s@samsung.com), Navjot Dhaliwal (navjot.tu@gmail.com), Nirmal Pandey (np@samsung.com), and Ukanti Chandu (chandu.ukanti@samsung.com).

**STUDENT INFO.**

By name or email By degree By specialization

Email Id All All

Search bar

**Akash Goel**  
akash.g@samsung.com

**Maninder Arora**  
maninder.s@samsung.com

**Navjot Dhaliwal**  
navjot.tu@gmail.com

**Nirmal Pandey**  
np@samsung.com

**Ukanti Chandu**  
chandu.ukanti@samsung.com

Main View









1:29 AM

STUDENT INFO.

By name or email
By degree
By specialization

Name
All
All



**Akash Goel**  
akash.g@samsung.com

**Maninder Arora**  
maninder.s@samsung.com

**Navjot Dhaliwal**  
navjot.tu@gmail.com

**Nirmal Pandey**  
np@samsung.com


Add Student


Delete Students

Menu-Items

Android status bar icons: Mail, USB, Android robot, SIM card, Signal, Battery, 1:18 AM

### STUDENT INFO.

By name or email    By degree    By specialization

Name    All    All

u

**Ukanti Chandu**

chandu.ukanti@samsung.com

---

Search by name

STUDENT INFO.

By name or email      By degree      By specialization

Email Id      All      All

n

**Navjot Dhaliwal**  
navjot.tu@gmail.com

**Nirmal Pandey**  
np@samsung.com

Search by email id



✉ 🔄 🤖 📶 🚫 🔋 1:16 AM

**STUDENT INFO.**

**Name of Student**  
Navjot Singh Dhaliwal

**Degree**  
B.E.

**Specialization**  
Computer Science

**Father's Name**  
Balwinder Singh

**Mother's Name**  
Kuldeep Kaur

**Birth Date**  
14-12-1988

**EmailId**  
navjot.tu@gmail.com



Details of a Student








1:30 AM

STUDENT INFO.

**Name of Student**  
Navjot Singh Dhaliwal

**Degree**  
B.E.

**Specialization**  
Computer Science

**Father's Name**  
Balwinder Singh

**Mother's Name**  
Kuldeep Kaur

**Birth Date**  
14-12-1988

EDIT

DELETE

Menu-Options

STUDENT INFO.

**First Name\***

Navjot

**Middle Name**

Singh

**Last Name\***

Dhaliwal

B.E.

Comp. Sc.

**Father Name\***

Add a Student

3G 10:04 AM

Student Info.

First Name\*

Navjot

Middle Name

Singh

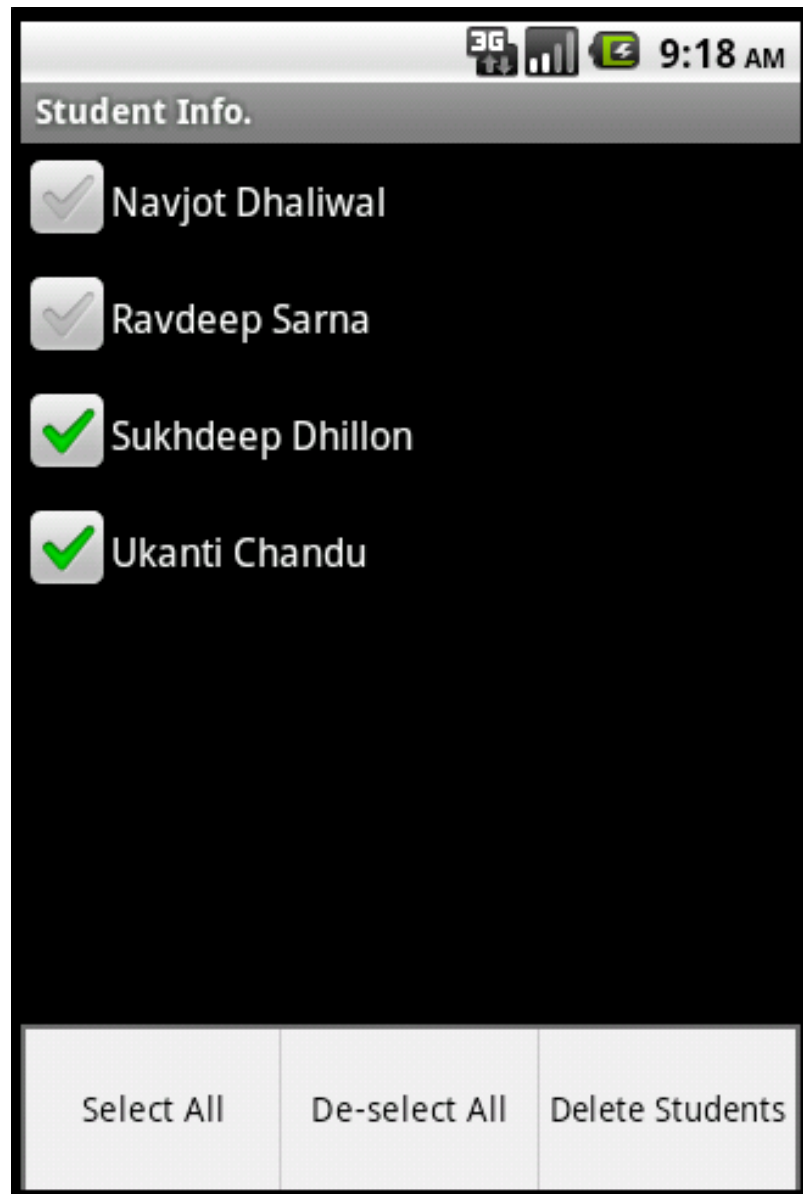
Last Name\*

Dhaliwal

B.E.

DONE

Edit a Student



Delete Students



## **Work Program**

The first 4 weeks of my training program were scheduled for class room sessions. The first week, there was brush up of Java skills. Next three weeks went to understanding and implementing demo android applications. For every classroom session there was a demo application to be developed. We covered the basic concepts of android like:

- Activities
- Services
- Intent
- Layouts
- Widgets
- Shared Preferences
- SQLite
- Adapters
- Broadcast Receiver
- Notification

After developing demo applications using above concepts, Student Information project started and it lasted for 2 weeks. I got help from my mentor and online API [developer.android.com](http://developer.android.com) site. The concepts used in this application are Activities, Intent, Layouts, Adapters, SQLite database.

### Test cases

Test case	Expected Result	Result
Searching by name or email id when list is empty	The list should remain empty	Normal Operation
Deleting students when list is empty	The delete student layout should be empty list	Normal Operation
Search student by name	Instant search for first names which are like the entered text	Normal Operation
Search student by email id	Instant search for email id which are like the entered text	Normal Operation
Different combinations of all 3 spinner dialogs, when using searching. Ex- Name starting with 'n', Degree is 'BE', Specialization is 'Comp. Sc'.	Should show the list of all students whose names starts with 'n', degree is 'BE', with specialization 'Comp Sc'	Normal Operation
While adding a student, the required fields are left blank or filled with inappropriate data types or email id is filled, which matches with email id of another student.	The student should not be added and a Toast text be displayed to tell the user which field has issues.	Normal Operation
While editing a student, the required fields are left blank or filled with inappropriate data types or email id is filled, which matches with email id of another student.	The student information should not be edited and a Toast text be displayed to tell the user which field has issues.	Normal Operation
Delete student/students	After being deleted, the student should not be displayed in the main list as well as the searched lists.	Normal Operation

### Conclusion and Future Scope

This application saves the students in device memory only. It could be extended to save the database on a server and insert/delete/select the record from the server. Also a photograph of the student could be saved in the database and displayed.

## References

<http://developer.android.com/reference/android/widget/ListView.html>

<http://developer.android.com/reference/android/widget/BaseAdapter.html>

<http://developer.android.com/reference/android/database/sqlite/SQLiteDatabase.html>

<http://developer.android.com/reference/android/widget/DatePicker.html>

<http://developer.android.com/reference/android/widget/Spinner.html>

<http://developer.android.com/reference/android/widget/EditText.html>

<http://developer.android.com/reference/android/widget/TextView.html>

<http://developer.android.com/reference/android/view/Menu.html>

<http://developer.android.com/reference/android/content/res/Resources.html>

## CONTACT TRACKER

Contact Tracker is an Android application that provides the following features :

- Option for calling a contact
- Maintaining Call and SMS log
- Sending a message to a contact
- Fetching a contact from default Contact List

### ContactsContract API

The contract between the contacts provider and applications. Contains definitions for the supported URIs and columns. These APIs supersede ContactsContract.Contacts.

#### Overview

ContactsContract defines an extensible database of contact-related information. Contact information is stored in a three-tier data model:

- A row in the ContactsContract.Data table can store any kind of personal data, such as a phone number or email addresses. The set of data kinds that can be stored in this table is open-ended. There is a predefined set of common kinds, but any application can add its own data kinds.
- A row in the ContactsContract.RawContacts table represents a set of data describing a person and associated with a single account (for example, one of the user's Gmail accounts).
- A row in the ContactsContract.Contacts table represents an aggregate of one or more RawContacts presumably describing the same person. When data in or associated with the RawContacts table is changed, the affected aggregate contacts are updated as necessary.

Other tables include:

- [ContactsContract.Groups](#), which contains information about raw contact groups such as Gmail contact groups. The current API does not support the notion of groups spanning multiple accounts.
- [ContactsContract.StatusUpdates](#), which contains social status updates including IM availability.
- [ContactsContract.AggregationExceptions](#), which is used for manual aggregation and disaggregation of raw contacts
- [ContactsContract.Settings](#), which contains visibility and sync settings for accounts and groups.
- [ContactsContract.SyncState](#), which contains free-form data maintained on behalf of sync adapters
- [ContactsContract.PhoneLookup](#), which is used for quick caller-ID lookup

## **ContactsContract.Data**

### Class Overview

Constants for the data table, which contains data points tied to a raw contact. Each row of the data table is typically used to store a single piece of contact information (such as a phone number) and its associated metadata (such as whether it is a work or home number).

### **Data kinds**

Data is a generic table that can hold any kind of contact data. The kind of data stored in a given row is specified by the row's [MIMETYPE](#) value, which determines the meaning of the generic columns [DATA1](#) through [DATA15](#). For example, if the data kind is [Phone.CONTENT\\_ITEM\\_TYPE](#), then the column [DATA1](#) stores the phone number, but if the data kind is [Email.CONTENT\\_ITEM\\_TYPE](#), then [DATA1](#) stores the email address. Sync adapters and applications can introduce their own data kinds.

ContactsContract defines a small number of pre-defined data kinds, e.g. [ContactsContract.CommonDataKinds.Phone](#), [ContactsContract.CommonDataKinds.Email](#) etc. As a convenience, these classes define data kind specific aliases for DATA1 etc. For example, [Phone.NUMBER](#) is the same as [Data.DATA1](#).

[DATA1](#) is an indexed column and should be used for the data element that is expected to be most frequently used in query selections. For example, in the case of a row representing email addresses [DATA1](#) should probably be used for the email address itself, while [DATA2](#) etc can be used for auxiliary information like type of email address.

By convention, [DATA15](#) is used for storing BLOBs (binary data).

The sync adapter for a given account type must correctly handle every data type used in the corresponding raw contacts. Otherwise it could result in lost or corrupted data.

Similarly, you should refrain from introducing new kinds of data for an other party's account types. For example, if you add a data row for "favorite song" to a raw contact owned by a Google account, it will not get synced to the server, because the Google sync adapter does not know how to handle this data kind. Thus new data kinds are typically introduced along with new account types, i.e. new sync adapters.

## Batch operations

Data rows can be inserted/updated/deleted using the traditional [insert\(Uri, ContentValues\)](#), [update\(Uri, ContentValues, String, String\[\]\)](#) and [delete\(Uri, String, String\[\]\)](#) methods, however the newer mechanism based on a batch of [ContentProviderOperation](#) will prove to be a better choice in almost all cases. All operations in a batch are executed in a single transaction, which ensures that the phone-side and server-side state of a raw contact are always consistent. Also, the batch-based approach is far more efficient: not only are the database operations faster when executed in a single transaction, but also sending a batch of commands to the content provider saves a lot of time on context switching between your process and the process in which the content provider runs.

The flip side of using batched operations is that a large batch may lock up the database for a long time preventing other applications from accessing data and potentially causing ANRs ("Application Not Responding" dialogs.)

To avoid such lockups of the database, make sure to insert "yield points" in the batch. A yield point indicates to the content provider that before executing the next operation it can commit the changes that have already been made, yield to other requests, open another transaction and continue processing operations. A yield point will not automatically commit the transaction, but only if there is another request waiting on the database. Normally a sync adapter should insert a yield point at the beginning of each raw contact operation sequence in the batch.

## **ContactsContract.RawContacts**

### Class Overview

Constants for the raw contacts table, which contains one row of contact information for each person in each synced account. Sync adapters and contact management apps are the primary consumers of this API.

### **Aggregation**

As soon as a raw contact is inserted or whenever its constituent data changes, the provider will check if the raw contact matches other existing raw contacts and if so will aggregate it with those. The aggregation is reflected in the [ContactsContract.RawContacts](#) table by the change of the [CONTACT\\_ID](#) field, which is the reference to the aggregate contact.

Changes to the structured name, organization, phone number, email address, or nickname trigger a re-aggregation.

## **ContactsContract.Contacts**

### **Class Overview**

Constants for the contacts table, which contains a record per aggregate of raw contacts representing the same person.

### **Operations**

#### **Insert**

A Contact cannot be created explicitly. When a raw contact is inserted, the provider will first try to find a Contact representing the same person. If one is found, the raw contact's [CONTACT\\_ID](#) column gets the \_ID of the aggregate Contact. If no match is found, the provider automatically inserts a new Contact and puts its \_ID into the [CONTACT\\_ID](#) column of the newly inserted raw contact.

#### **Update**

Only certain columns of Contact are modifiable: [TIMES\\_CONTACTED](#), [LAST\\_TIME\\_CONTACTED](#), [STARRED](#), [CUSTOM\\_RINGTONE](#), [SEND\\_TO\\_VOICEMAIL](#). Changing any of these columns on the Contact also changes them on all constituent raw contacts.

#### **Delete**

Be careful with deleting Contacts! Deleting an aggregate contact deletes all constituent raw contacts. The corresponding sync adapters will notice the deletions of their respective raw contacts and remove them from their back end storage.

#### **Query**

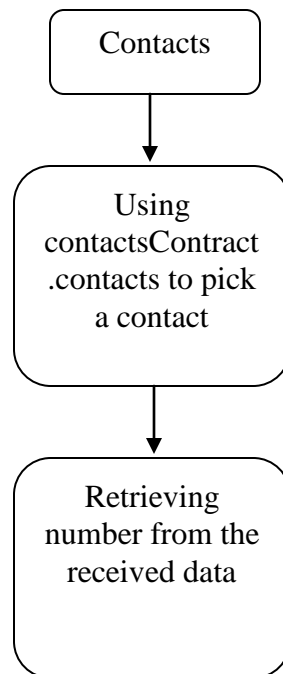
- If you need to read an individual contact, consider using [CONTENT\\_LOOKUP\\_URI](#) instead of [CONTENT\\_URI](#).
- If you need to look up a contact by the phone number, use [PhoneLookup.CONTENT\\_FILTER\\_URI](#), which is optimized for this purpose.

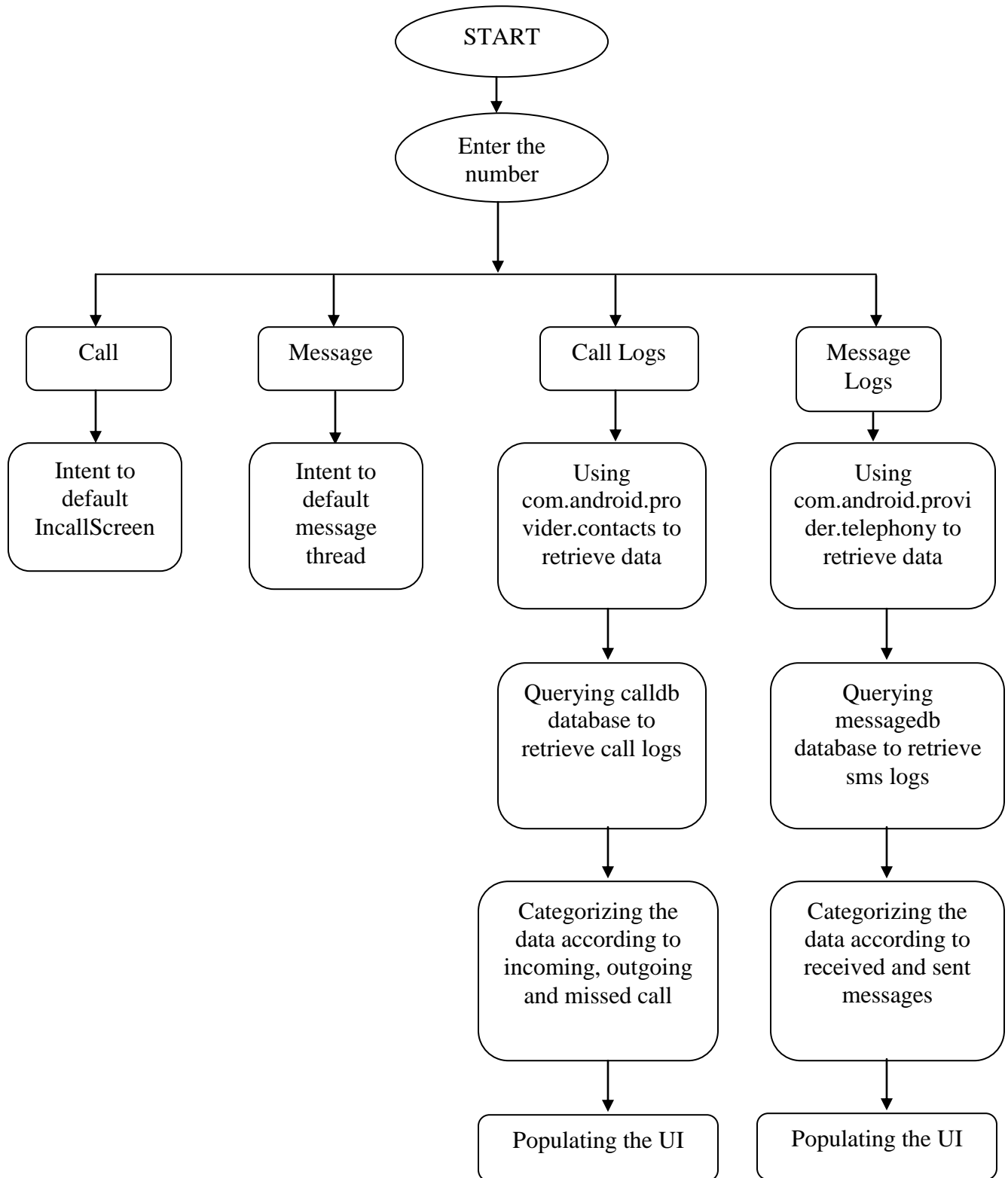


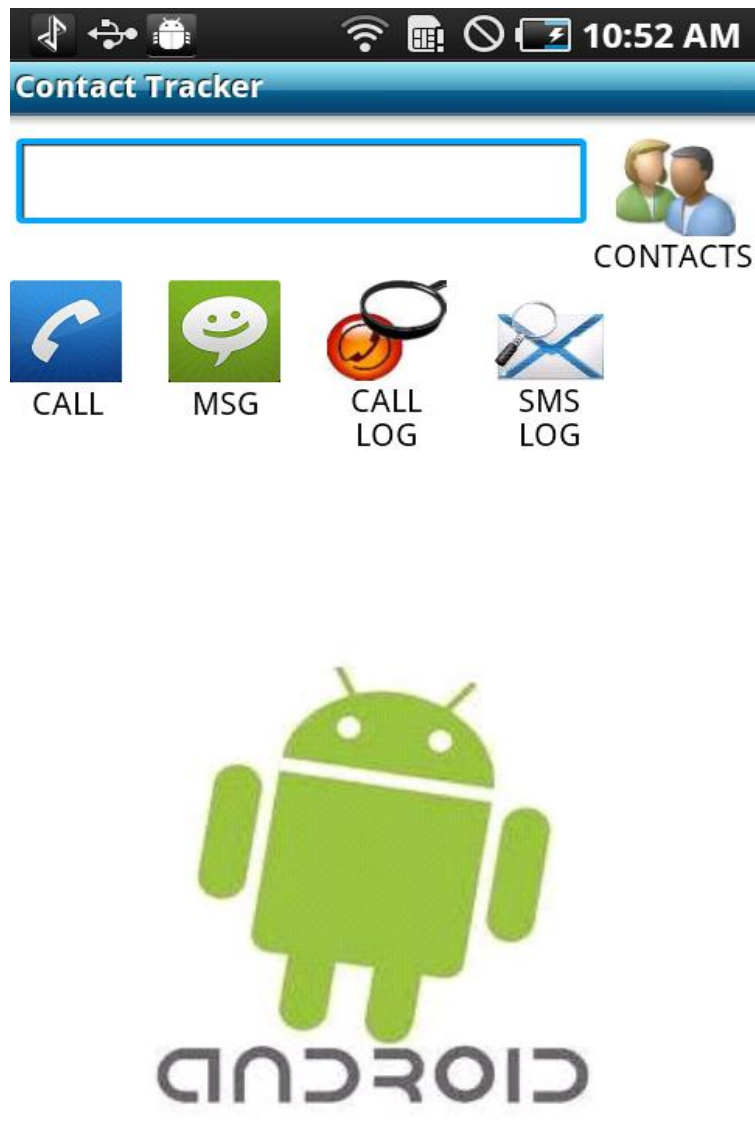
- If you need to look up a contact by partial name, e.g. to produce filter-as-you-type suggestions, use the [CONTENT\\_FILTER\\_URI](#) URI.
- If you need to look up a contact by some data element like email address, nickname, etc, use a query against the [ContactsContract.Data](#) table. The result will contain contact ID, name etc.

## APPLICATION FEATURES

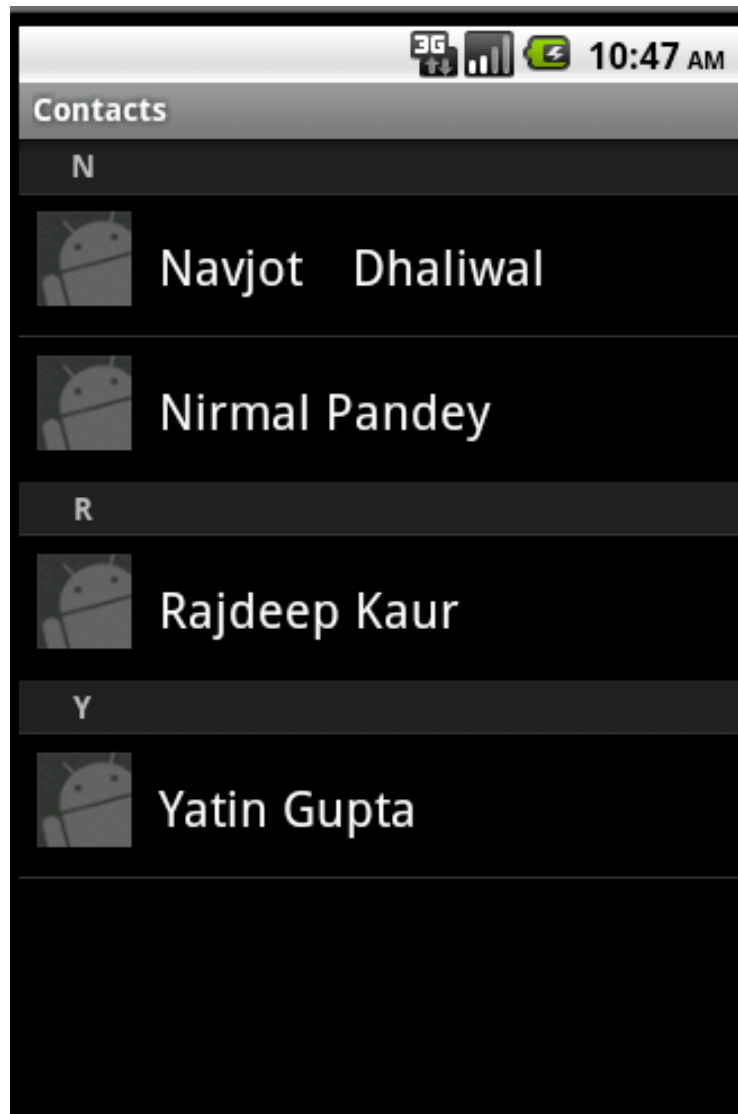
### Flow Diagram



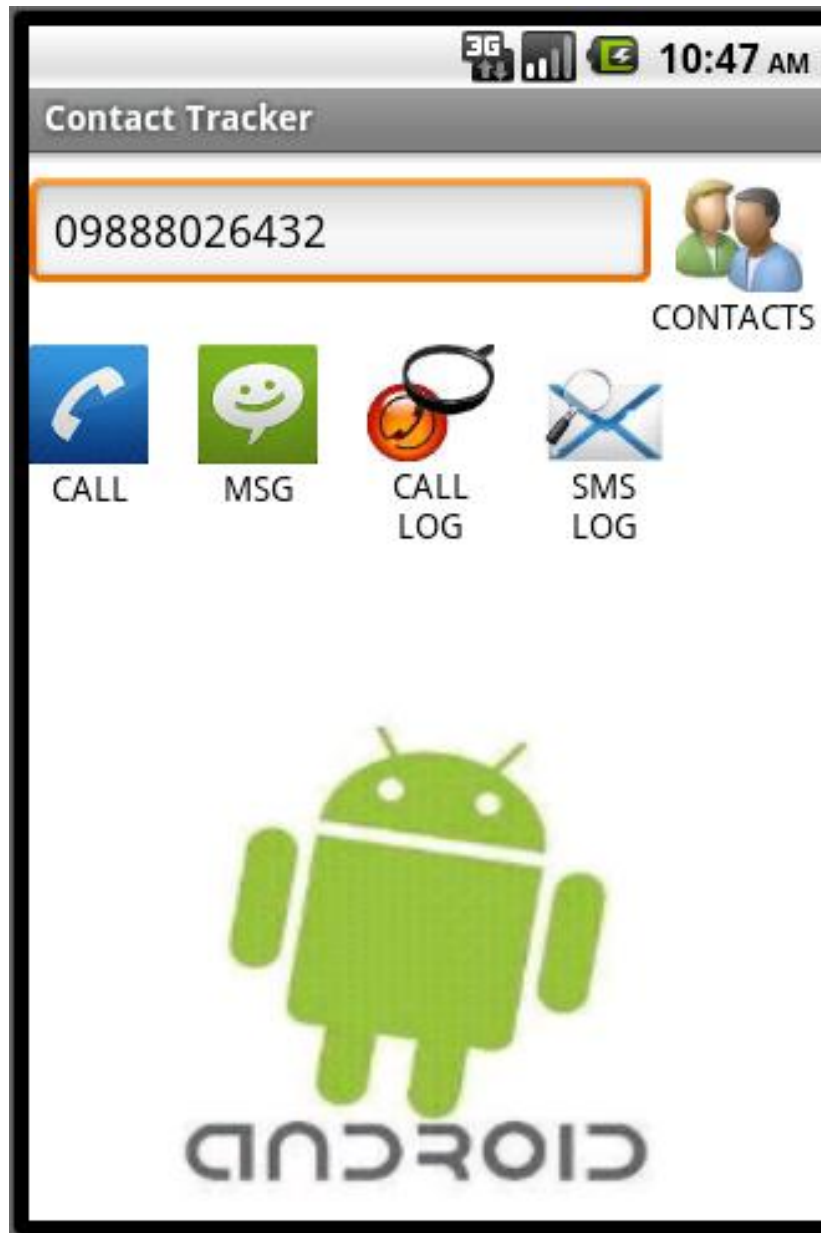




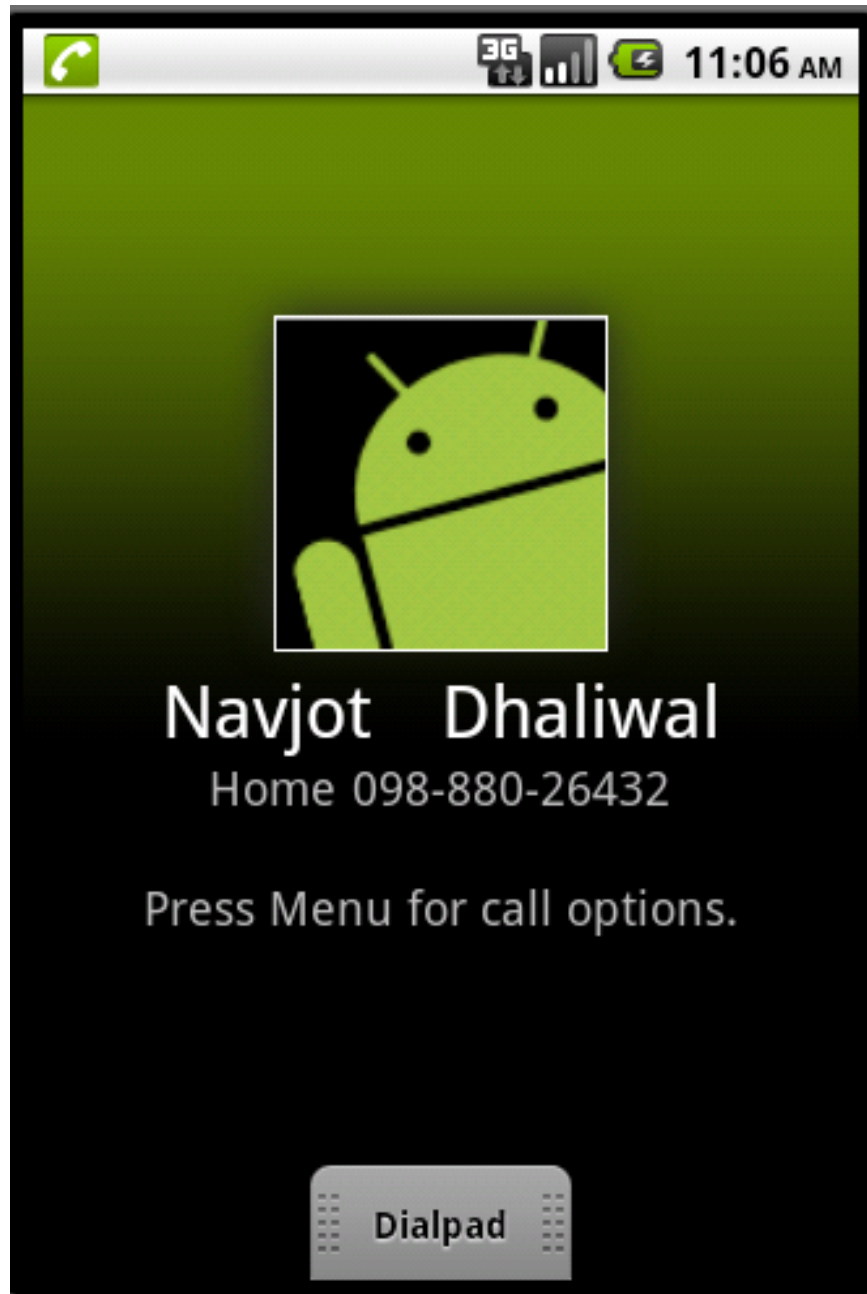
Main Screen



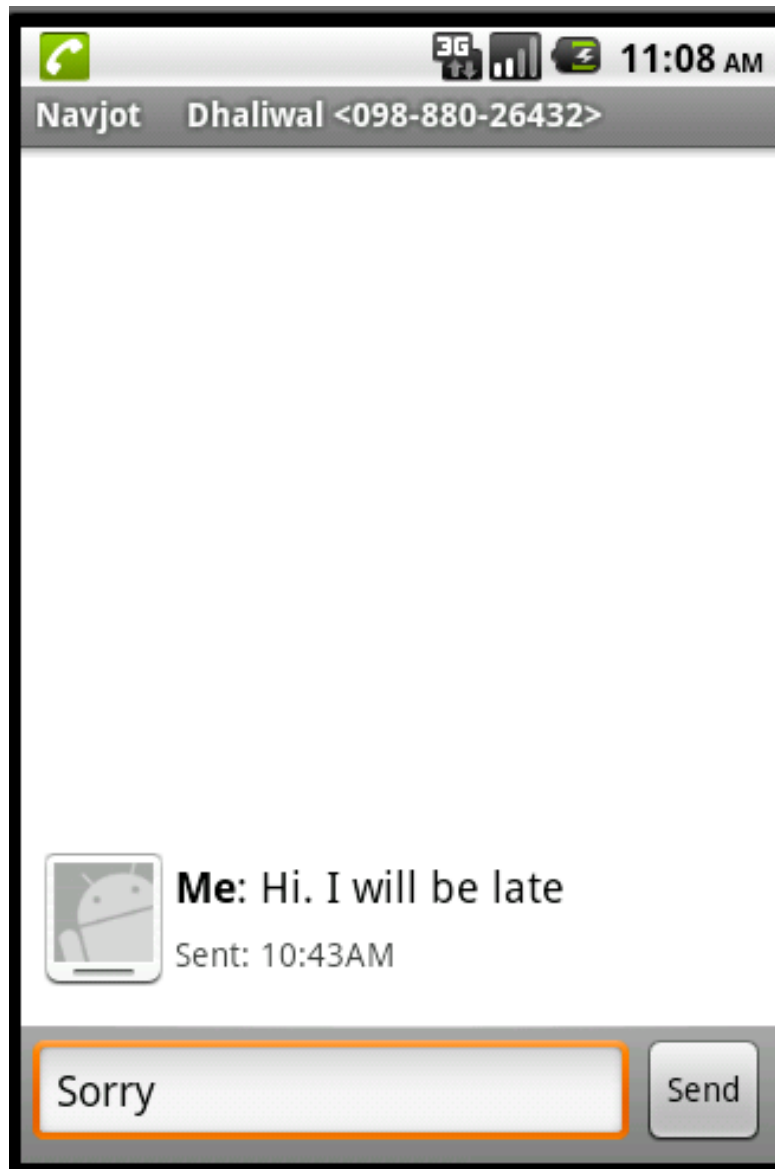
Accessing contacts from default contacts database



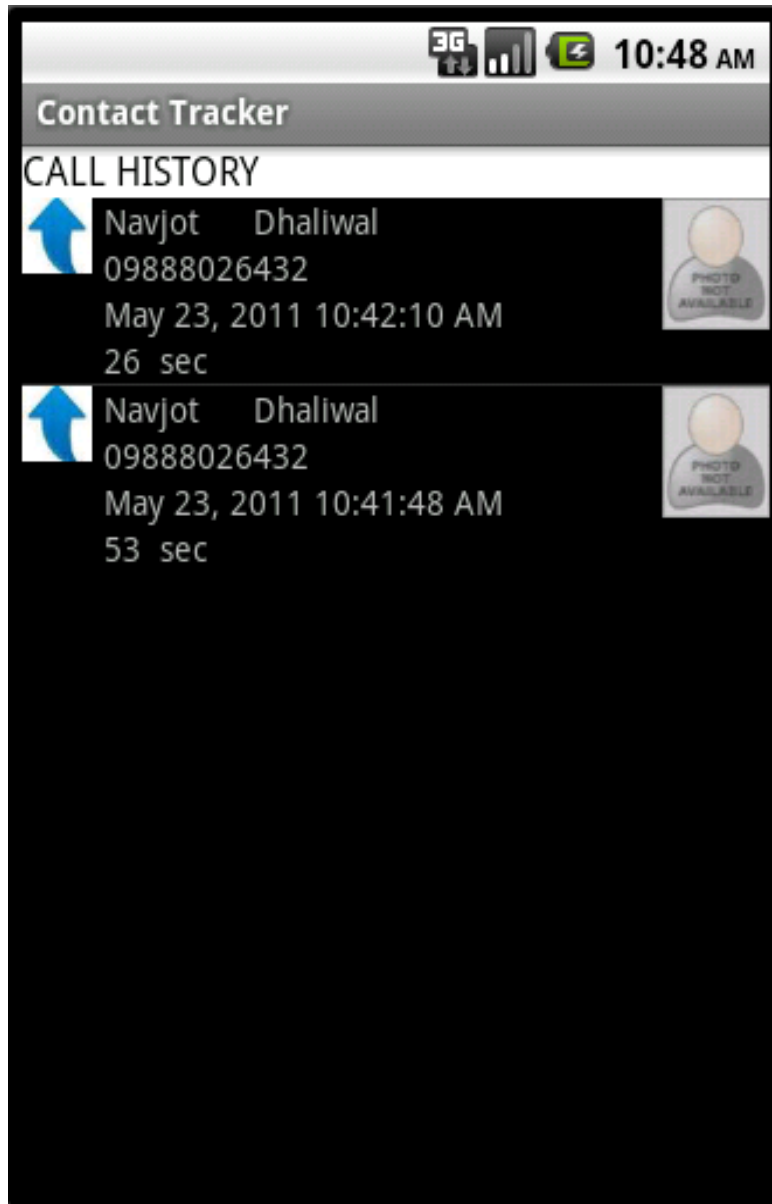
After selecting a contact



Call

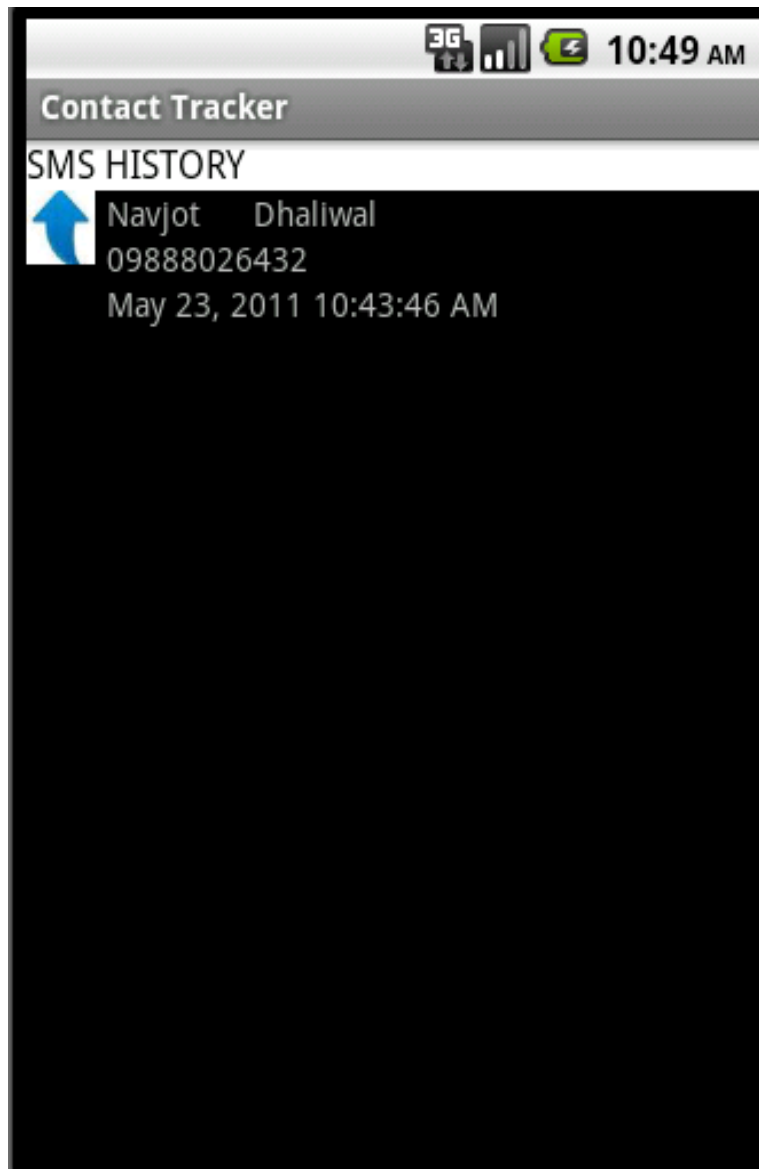


Messaging

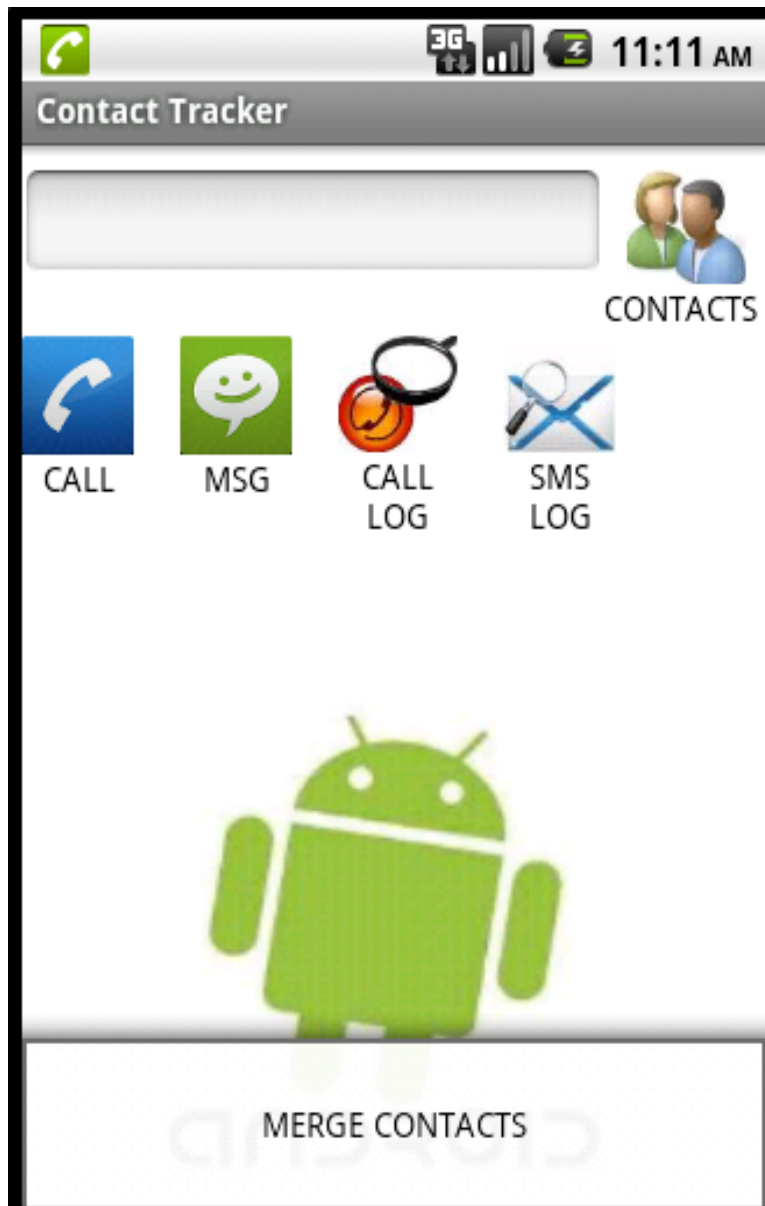


Call Log

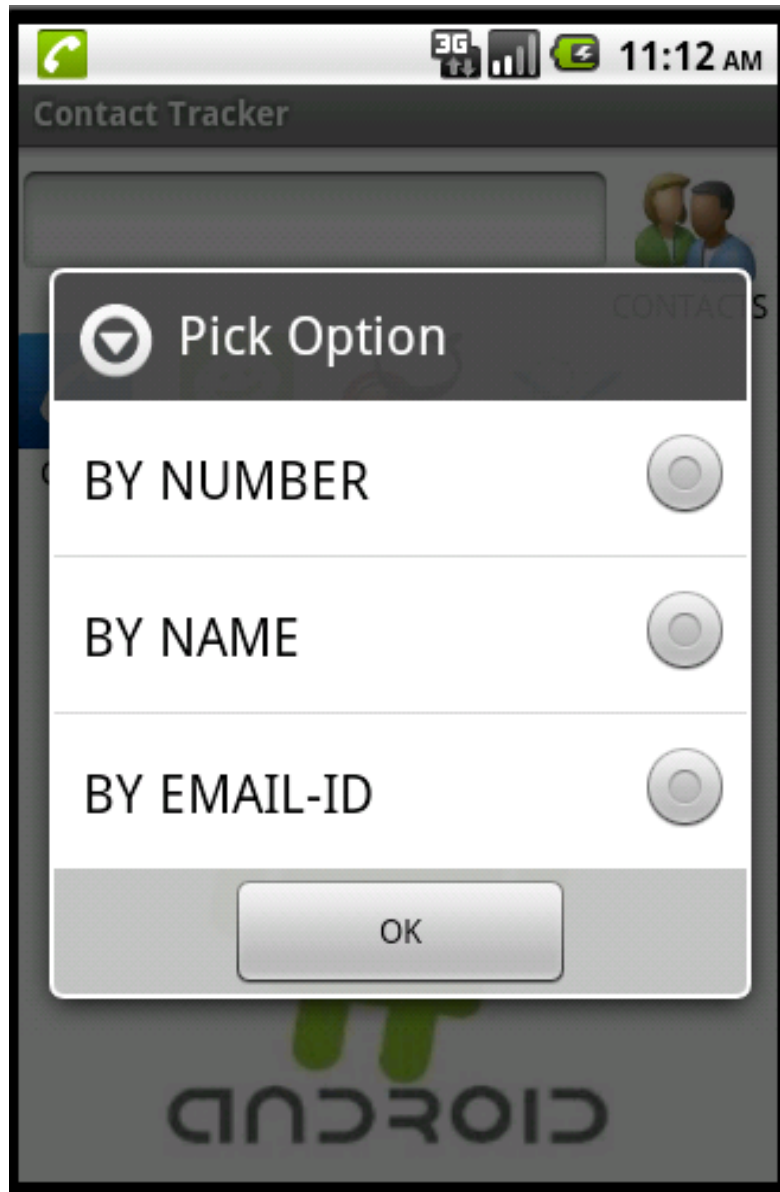




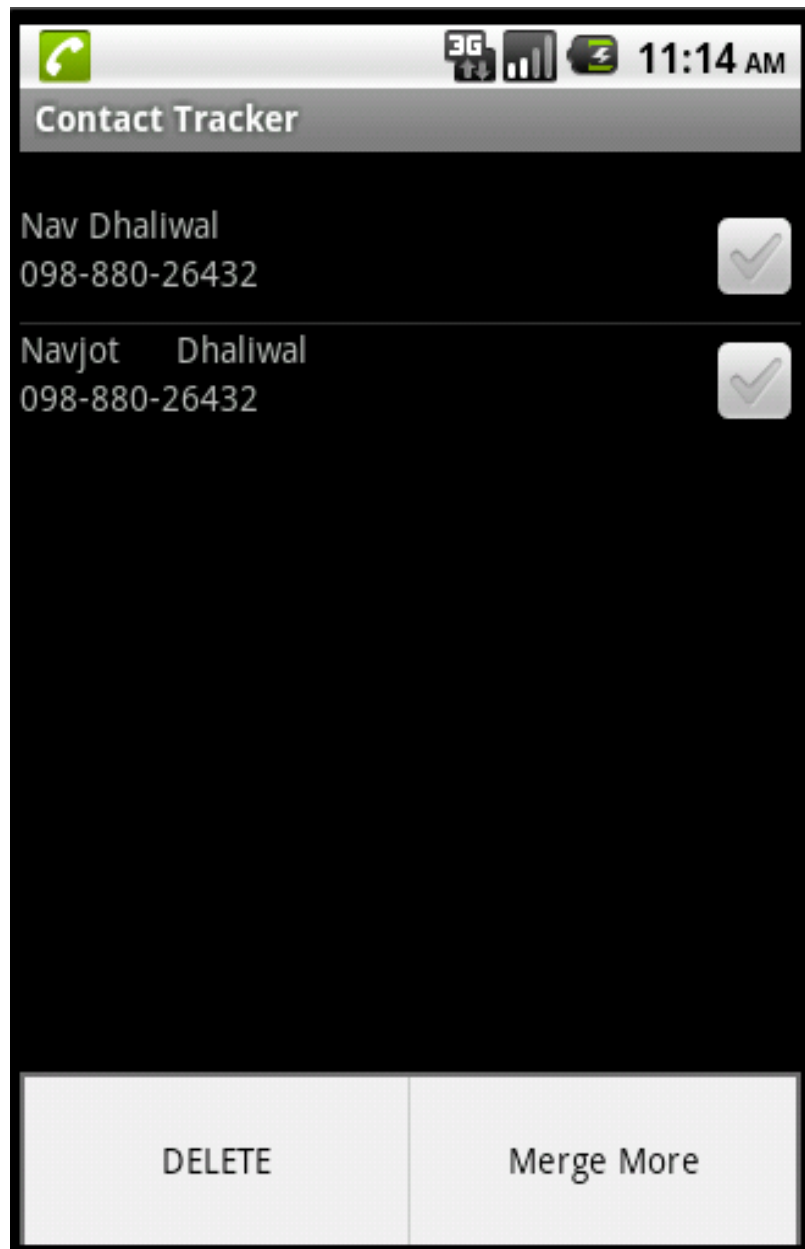
Sms Log



Merging Contacts Option



Merge By Options



Merge By Number

## **Work Program**

After Student Information project, this application was assigned to develop. This also took around 2 weeks. The main android concepts used were:

- Activities
- Intents
- Views
- Layouts
- Content Providers
- Adapters

This project was given to familiarize me with calling and messaging basics. I understood the use of in-built databases.

### Test Cases

Test Case	Expected Result	Result
Trying to call/message/call log/msg log with empty field	Toast text displaying to enter a number	Normal Operation
If we call/msg a number, which is not saved in contacts, then check logs. Save that number and check the logs.	The log should be updated with the contacts name and other information.	Normal Operation as everytime the logs fetch the information from the database.
Merge Option	While merging, application asks to delete a contact. This contact should be deleted from contact database also.	Normal Operation. Check the contacts after deleting one of the entries.

### Conclusion and Future Scope

This application is really helpful when we need to view all the logs of a particular contact or a number.

In this application voice enabled messaging and logs could be displayed using Google's Speech Input and Voice Recognition API.

### References

<http://developer.android.com/reference/android/provider/ContactsContract.Contacts.html>

<http://developer.android.com/reference/android/content/ContentProvider.html>

## DEMO APPLICATION FOR NOTIFICATION

### Introduction to Notifications

What are **Notifications**? The name itself implies their functionality. They are a way of alerting a user about an event that he needs to be informed about or even take some action on getting that information.

Notification on Android can be done in any of the following ways:

- Status Bar Notification
- Vibrate
- Flash lights
- Play a sound

From the Notification, you can allow the user to launch a new activity as well. Now we will look at status bar notification as this can be easily tested on the emulator.

To create a status bar notification, you will need to use two classes: Notification and NotificationManager.

- **Notification** – defines the properties of the status bar notification like the icon to display, the text to display when the notification first appears on the status bar and the time to display.
- **NotificationManager** – is an android system service that executes and manages all notifications. Hence you cannot create an instance of the NotificationManager but you can retrieve a reference to it by calling the getSystemService() method.

Once you procure this handle, you invoke the `notify()` method on it by passing the notification object created.

So far, you have all the information to display on the status bar. However, when the user clicks the notification icon on the status bar, what detailed information should you show the user? This is yet to be created. This is done by calling the method `setLatestEventInfo()` on the notification object. What needs to be passed to this method, we will see with an example.

The code is explained below:

Step 1: Procure a handle to the `NotificationManager`:

```
private NotificationManager mNotificationManager;  
...  
mNotificationManager =  
    (NotificationManager) getSystemService(NOTIFICATION_SERVICE);
```

Step 2: Create a notification object along with properties to display on the status bar

```
final Notification notifyDetails =  
new Notification(R.drawable.android, "New Alert, Click  
Me!", System.currentTimeMillis());
```

Step 3: Add the details that need to get displayed when the user clicks on the notification. In this case, I have created an intent to invoke the browser to show the website <http://www.android.com>



```

Context context = getApplicationContext();

CharSequence contentTitle = "Notification Details...";

CharSequence contentText = "Browse Android Official Site by clicking me";
Intent notifyIntent
= new Intent(android.content.Intent.ACTION_VIEW,Uri.parse("http://www.android.com"
));

PendingIntent intent =
PendingIntent.getActivity(SimpleNotification.this, 0,
notifyIntent, android.content.Intent.FLAG_ACTIVITY_NEW_TASK);
notifyDetails.setLatestEventInfo(context, contentTitle, contentText, intent);

```

Step 4: Now the stage is set. Notify.

```
mNotificationManager.notify(SIMPLE_NOTIFICATION_ID, notifyDetails);
```

Note that all of the above actions(except getting a handle to the NotificationManager) are done on the click of a button “Start Notification”. So all the details go into the `setOnClickListener()` method of the button.

Similarly, the notification, for the example sake is stopped by clicking a cancel notification button. And the code there is :

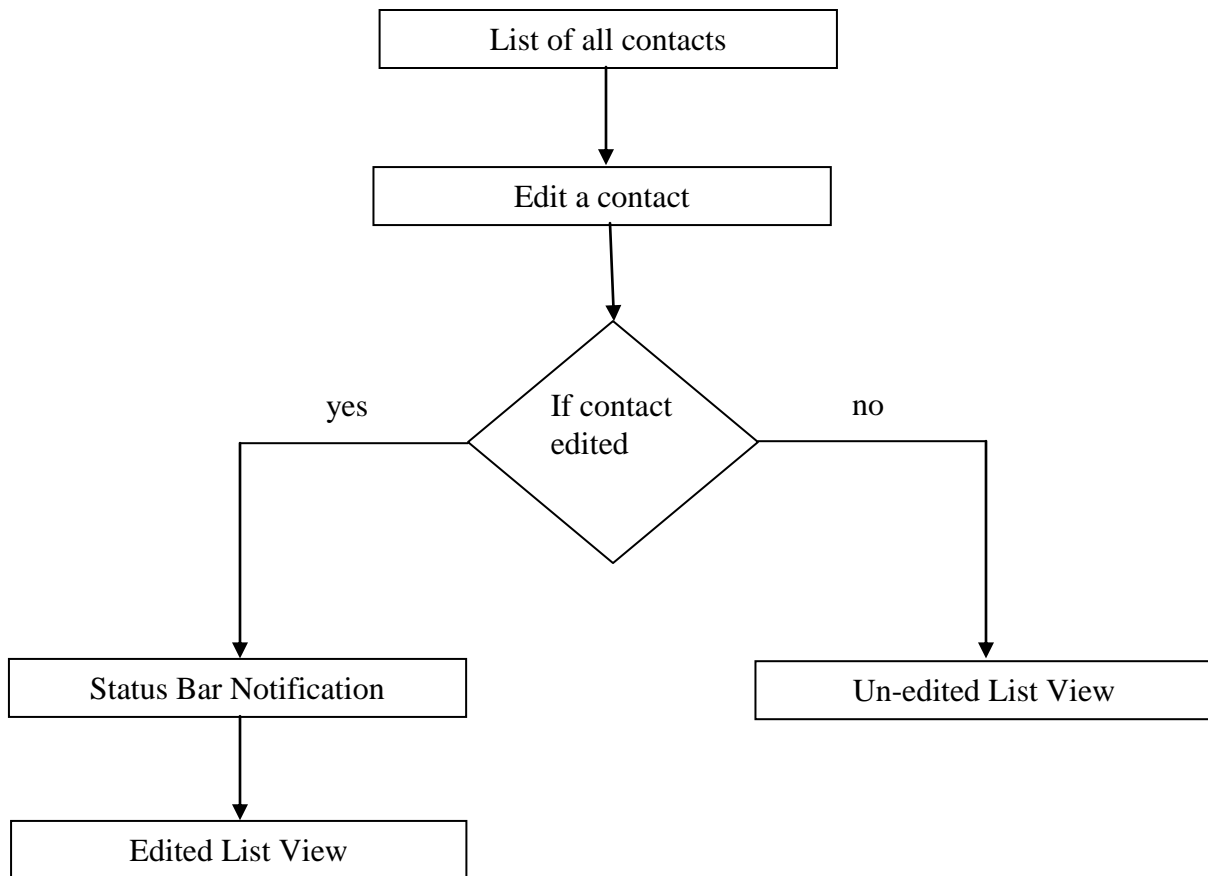
```
mNotificationManager.cancel(SIMPLE_NOTIFICATION_ID);
```

Now, you may realize that the constant `SIMPLE_NOTIFICATION_ID` becomes the way of controlling, updating, stopping a current notification that is started with the same ID.

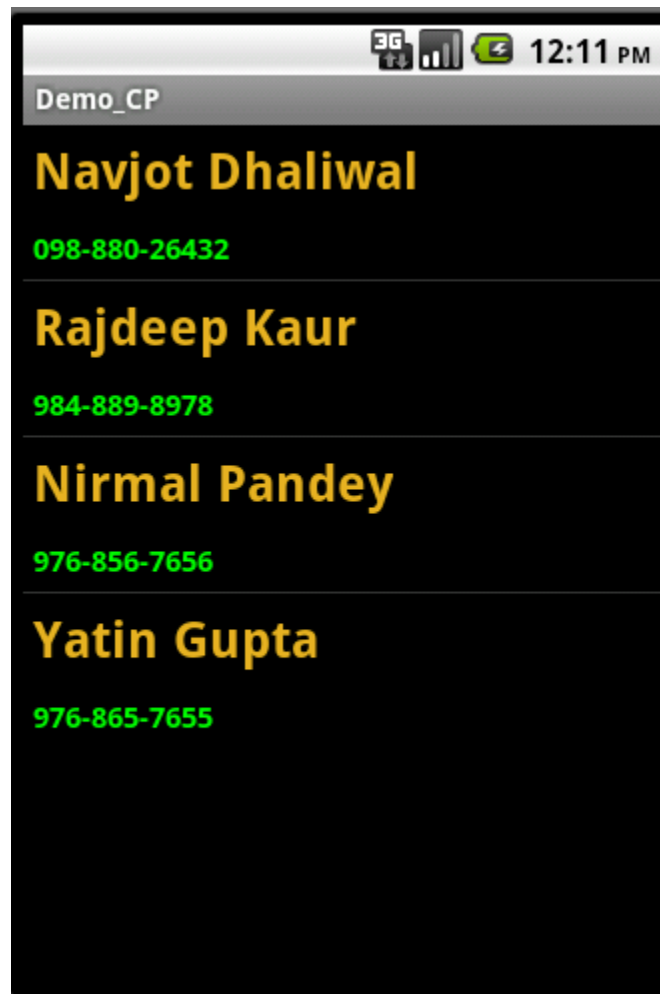
## THE APPLICATION

In this application, the first view is a ListView which shows all the phone contacts. By selecting any of these contacts, opens up the edit options of that contact. If there are any changes made to the contact, a status bar notification will be displayed.

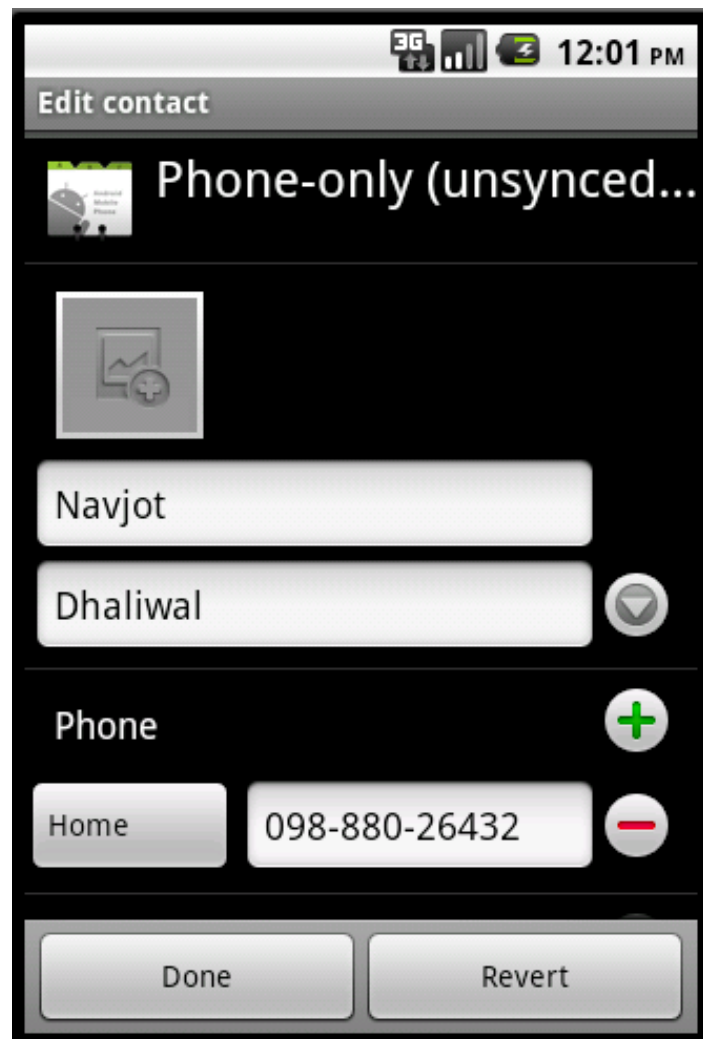
### Flow Diagram



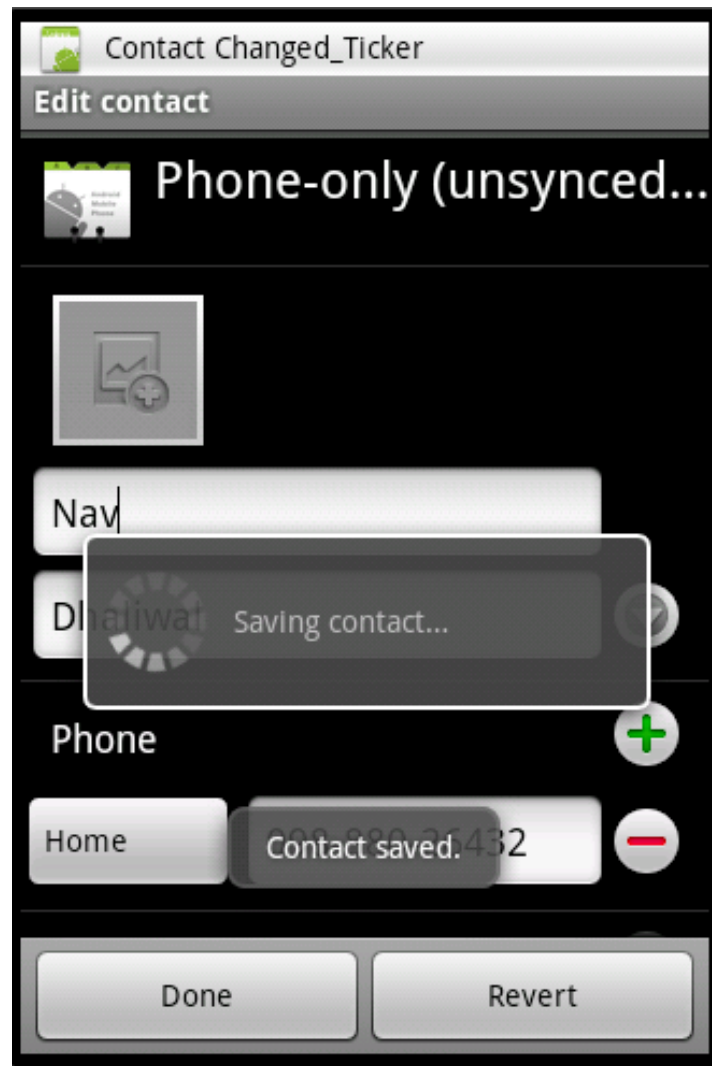
## Snap-Shots



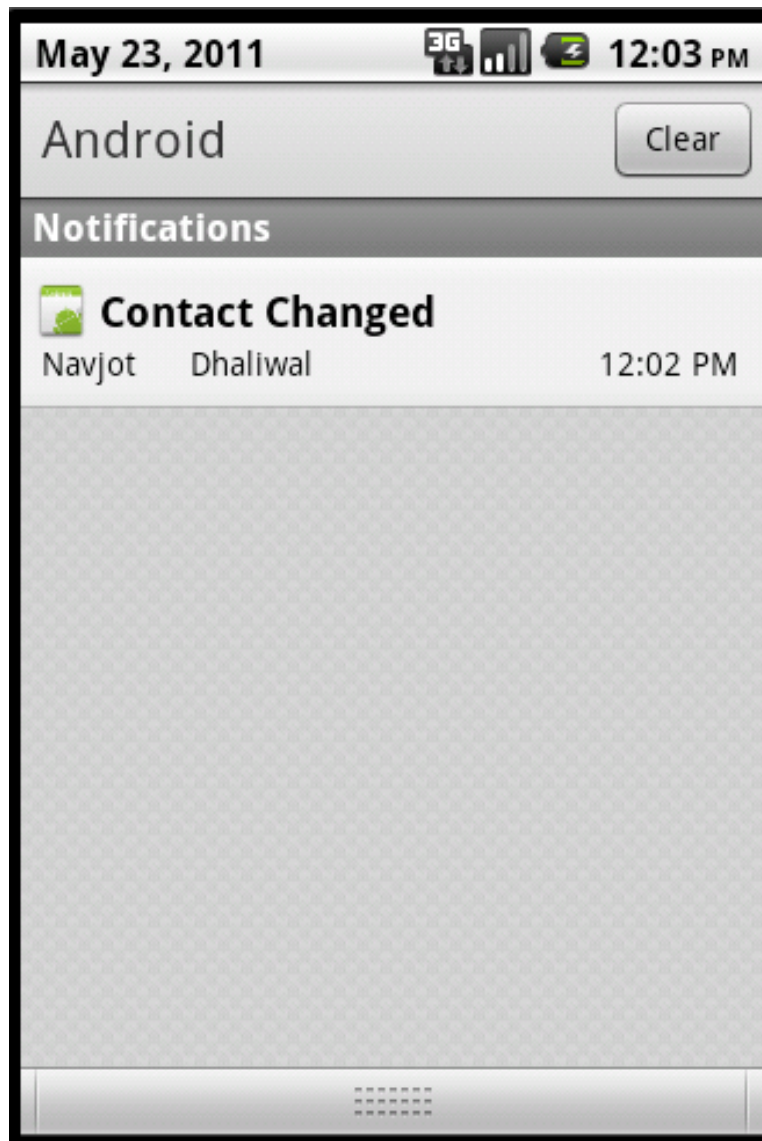
Main View



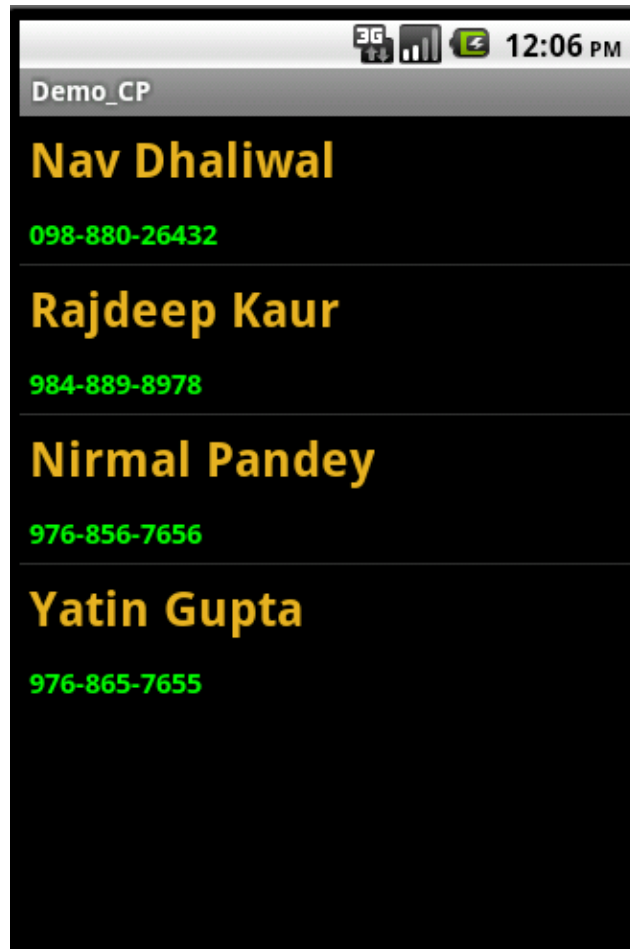
The Edit View



Contact Changed



Status Bar Notification



After Editing

### **Work Program**

After the initial 2 projects, I was assigned the Android Framework team. In the first week, I was given some demo applications to make, so that I understand the framework level work. This and the demo application on gestures were such applications.

### **References**

<http://developer.android.com/reference/android/app/Notification.html>

<http://developer.android.com/reference/android/app/NotificationManager.html>

## DEMO APPLICATION ON GESTURES

One of the most widespread changes seen in the use of touch screen devices in the last couple of years has been the adoption of finger gestures, such as swipes and flings. They allow the interaction between user and device to become very intuitive and natural feeling.

In this application, we are using a custom view and overriding the method `onDraw(canvas)`, to draw the image of puck at different points.

We use an object of `GestureDetector` to handle the gestures along with `GestureListener`. We override the methods in `GestureListener` interface. The methods are:

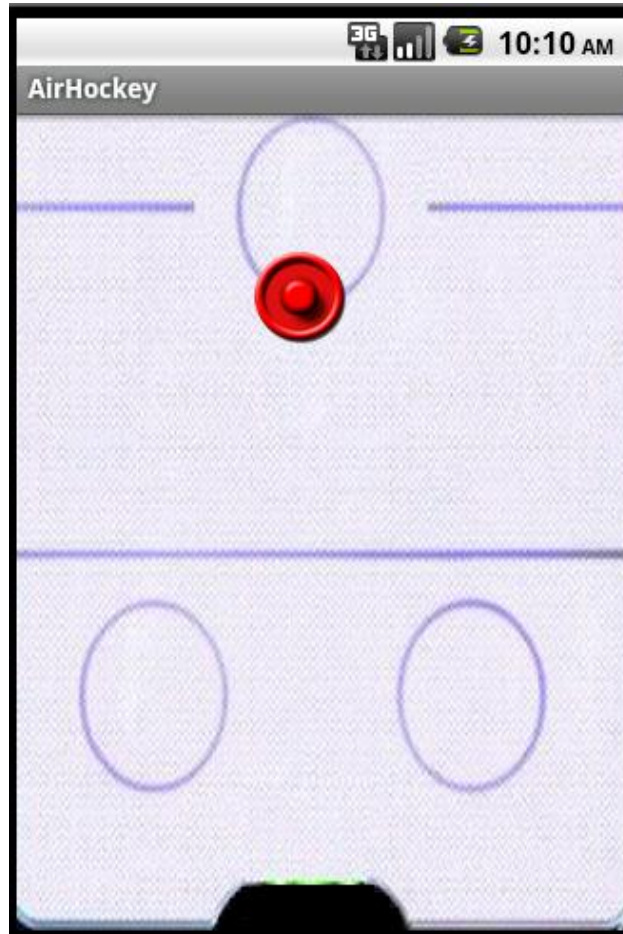
- `onDown`
- `onFling`
- `onLongPress`
- `onScroll`
- `onShowPress`
- `onSingleTapUp`
- `onDoubleTap`
- `onDoubleTapEvent`
- `onSingleTapConfirmed`

In `onFling` method, there are velocity arguments along x and y directions. Using the velocity in x and y direction, we calculate the x and y co-ordinates of the puck. These x and y co-ordinates are used in `onDraw()` method. In `onDraw()`, we call the `drawBitmap` method of canvas class. This method has following arguments:

- `Bitmap` – The image of puck.
- `Left` – The left corner of the image.
- `Top` – The top corner of the image.
- `Paint` – The paint object to be used.



We also have to check at every call of `onDraw()` that the image is not going outside the visible region. If it is too near to edges, we change the direction of velocity, thus changing the course of the puck. This creates an illusion that the puck returns after bouncing back from the edges, in accord with the laws of physics.



### The Main View

### **References**

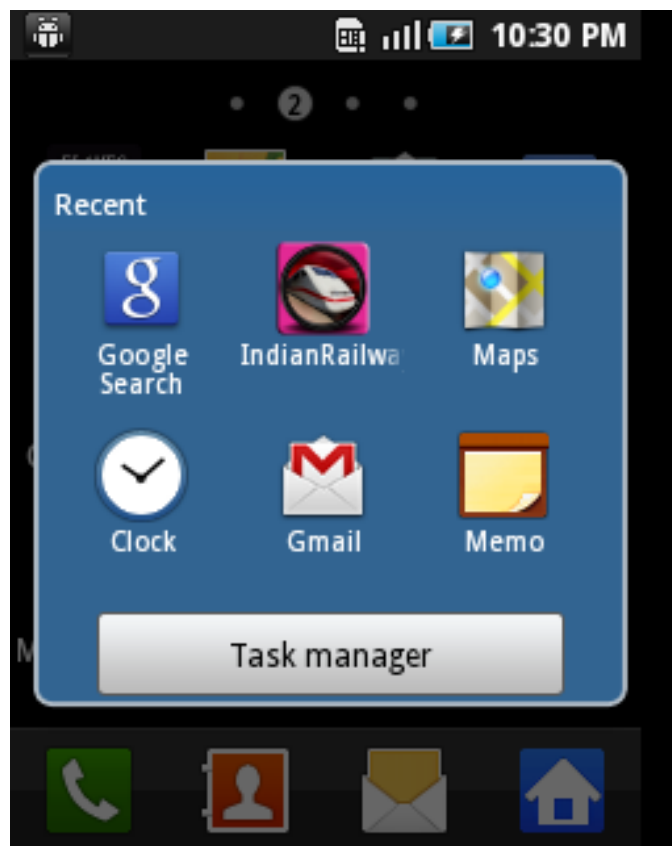
<http://developer.android.com/reference/android/view/GestureDetector.OnGestureListener.html>

## RECENTAPPLICATIONS

### Introduction

This application was developed specifically for T Mobile's Sidekick Device. There is an inbuilt application in android which shows the recent applications which were recently run on the android device. This application is shown when we long press the Home button.

The default application's main activity is "RecentApplicationsDialog". As the name clearly states that it is a dialog, with 6 recent applications shown as image buttons. At the bottom, there is button for Task Manager.



Default Recent Application Dialog

If we press any of the image buttons, that specific activity is launched. The main classes used are:

- **ActivityManager** - Interact with the overall activities running in the system.
- **PackageManager** - Class for retrieving various kinds of information related to the application packages that are currently installed on the device. You can find this class through `getPackageManager()`.
- **ActivityManager.RecentTaskInfo** - Information you can retrieve about tasks that the user has most recently started or visited.
- **ResolveInfo** - Information that is returned from resolving an intent against an `IntentFilter`. This partially corresponds to information collected from the `AndroidManifest.xml`'s `<intent>` tags.
- **ActivityInfo** - Information you can retrieve about a particular application activity or receiver. This corresponds to information collected from the `AndroidManifest.xml`'s `<activity>` and `<receiver>` tags.

First we make a list of all the recent tasks. When we call the method “`getRecentTasks`” for an “`ActivityManager`” object, it returns a list of objects of “`RecentTaskInfo`” class. For every object of “`RecentTaskInfo`”, we find the intent of that activity. From this intent, we find out the “`ResolveInfo`” object by calling the `PackageManager`’s `resolveActivity` method.

The object of `ResolveInfo` is used to find the title and icon of the corresponding activity. These attributes are then assigned to the image buttons.

## **APPLICATION FEATURES**

According to the requirements of T Mobile, Sidekick's Recent Applications are to be shown in a list instead of a dialog, and the list is to be slightly tilted. There is no inbuilt feature in android to have a tilted list view, so this feature is developed from scratch. The number of applications also is to be 8 and not 6.

Along with this, the first element and the last elements are to be set. The first is to be a lock screen shortcut and the last is to be referring to all apps.

When lock screen shortcut option is set to an application from the Settings, whenever the lock screen is unlocked, that particular application will be launched. Similarly, when all apps option is selected, it opens the launcher screen.

It also displays the assigned shortcuts to installed apps. The shortcuts are also set from the Settings and saved in a Database. The shortcut letters are shown on the far left side of the screen.

Due to this feature being Samsung proprietary, project work flow, components used and testing phase cannot be divulged.

### **Work Program**

The framework team was responsible for many of Sidekick's features including Recent Applications. Besides working on this feature, I also understood other features like, custom notifications and themes. Being in the android framework team, I also had the opportunity of working with some of the latest and upcoming devices.

## References

<http://developer.android.com/reference/android/app/ActivityManager.html>

<http://developer.android.com/reference/android/content/pm/PackageManager.html>

<http://developer.android.com/reference/android/app/ActivityManager.RecentTaskInfo.html>

<http://developer.android.com/reference/android/content/pm/ResolveInfo.html>