HCTL Open International Journal of Technology Innovations and Research (IJTIR)
http://ijtir.hctl.org
Volume 14, April 2015
e-ISSN: 2321-1814, ISBN (Print): 978-1-62951-946-3

An Android Based EZY Application for Calling, SMS & Email

Dimple Bajaj

dimplebajaj20785@gmail.com

Abstract

Introduced the main features of the Android. Highlighted the architecture of Android and gave a detailed description of the framework for making android applications from the developer's perspective. This paper could provide an insight as to how android applications are made. As an example a simple application which adds features to the Calling, SMS and E-mailing is provided as an instance to explain the working of android application components.

Keywords

Linux Kernel, Android System, Application Framework, Dalvik Virtual Machine

Introduction

Application framework defined the common structure of programs in the specific domain. Essentially, a framework is a component that can be reused, it set the architecture of applications and incorporated as a set of abstract classes and the cooperation of their instances. Android is an open source operating system based on Linux kernel and launched by Google. Android consists of a kernel based on the Linux kernel, with middleware, libraries and APIs written in C and application software running on an application framework which includes Java-compatible libraries based on Apache Harmony. Android uses the Dalvik virtual machine with just-in-time compilation to run Dalvik dex-code (Dalvik Executable), which is usually translated from Java bytecode.

Unlike PC operating system, mobile phone operating systems are constrained by their hardware, storage space, power dissipation and mobility conditions. Compared with the development of applications on PC, there are some different features of applications on mobile phone operating systems. This paper introduced the basic architecture and application framework of Android operating system, gives a detailed description of main structure of Android applications and the methods of developing applications based on Android application framework.

[1]DIT University, Dehradun, Uttarakhand, India.

Android Architecture

Android consists of a kernel based on the Linux kernel, with middleware, libraries and APIs written in C and application software running on an application framework which includes Java-compatible libraries based on Apache Harmony. Android uses the Dalvik virtual machine with just-in-time compilation to run Dalvik dex-code (Dalvik Executable), which is usually translated from Java bytecode.

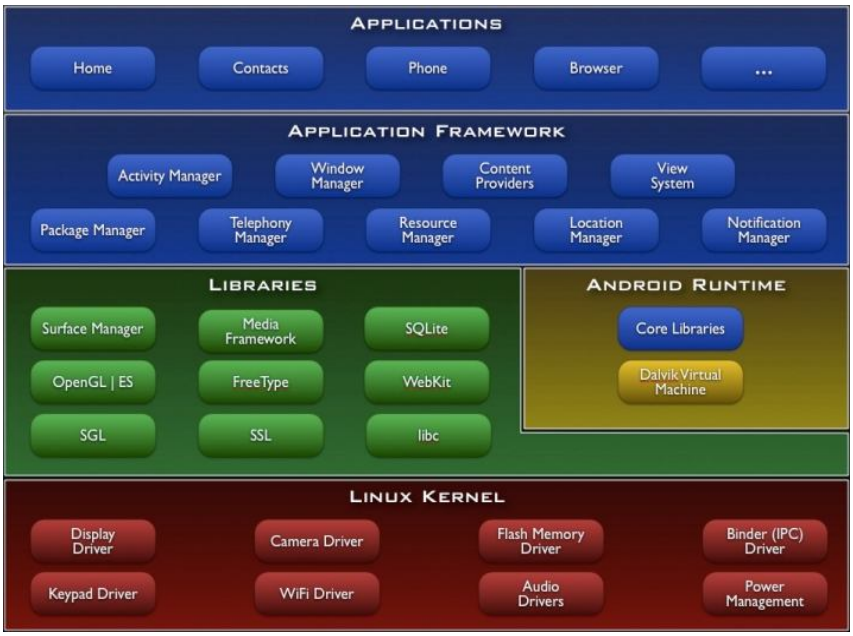

Figure 1: A sample line graph using colours which contrast well both on screen and on a black-and-white hardcopy

Applications

The application layer includes 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 capabilities (subject to security constraints enforced by the framework). This same mechanism allows components to be replaced by the user.

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 backstack

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's kernel is based on the Linux kernel and has further architecture changes by Google outside the typical Linux kernel development cycle. Android does not have a native X Window System nor does it support the full set of standard GNU libraries, and this makes it difficult to port existing Linux applications or libraries to Android.

Android Applications Architecture

Android applications are written in the Java programming language. The Android SDK tools compile the code – along with any data and resource files—into an Android package, an archive file with an .apk suffix. All the code in a single .apk file is considered to be one application and is the file that Android-powered devices use to install the application.

Android system implements the multiuser Linux System and each application has access to only those components which it requires to do its work. System assigns each application a unique user id and each process has its own virtual machine (VM), so that each application runs in isolation to other applications.

However, applications can share data with each other. This is implemented by two applications sharing the same user id and virtual machine to run their processes.

Android Application Components

Application Components are essential building blocks of an android application and define the overall application's behaviour. Each component is a different point through which the system can enter your application. However not all components are actual entry points for the user but each one exists as its own entity and plays a specific role. There are four types of application components. There are four types of application components: activities, services, content providers and broadcast receivers.

Activity provides a screen for an application to provide user interface. Each activity is given a window which either fills the screen or floats at the top of other windows. There can be many activities in an application as depending upon the features provided but there exists an activity which is called the "main" activity. It starts as soon as the application is launched and is responsible for starting all other activities to perform various functions. Each time a new activity starts, the system preserves the previous activity in a stack, "the back stack" after stopping it. Now when the user is done with the current activity, the back stack pops out the current activity and previous activity resumes.

A new activity is always created as a subclass of the class "activity". And in the subclass, various callback methods are implemented that the system calls when the activity transitions between various states of it lifecycle. Callback methods notify the current activity about changes in its state and provide the opportunity to perform specific work

that's appropriate to that state change. onCreate() and onPause() are important callback methods.

The user interface for an activity is provided by a hierarchy of views which are objects derived from the View class. Android also provides “Widgets” which are readymade views that provide elements for the screen. "Layouts" are views that are derived from View Group class and provide a unique layout model for its child views. The most common way to define a layout using views is with an XML layout file saved in your application resources.

1) ***Starting an Activity***: The system calls in an activity lifecycle are called in a sequence similar to a step pyramid. Firstly the activity is created and each callback method moves the activity state one step towards the top of the step pyramid. At the top of the pyramid the activity is running in the foreground. As the user begins to leave the activity, the system calls methods that move the activity down the pyramid.

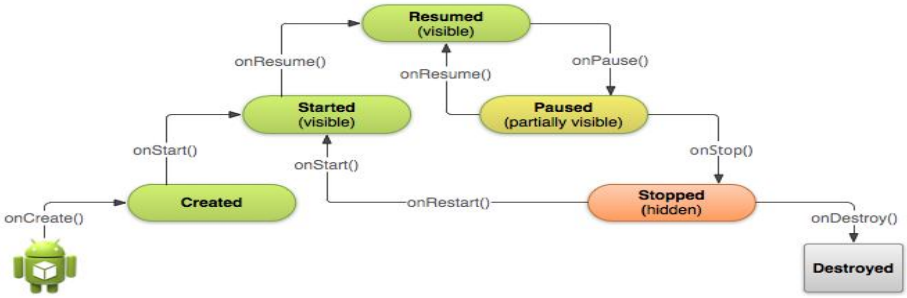

Figure 2: A sample diagram showing how to start an activity

In created state the instance of the activity is created. The next state is resumed, here the activity is in the foreground and the user can interact with it. In paused state, the activity is partially obscured by another activity i.e. the other activity that's in the foreground is semi-transparent or doesn't cover the entire screen. The paused activity does not receive user input and cannot execute any code. In stopped state, the activity is completely hidden and not visible to the user.

2) ***Pausing and Resuming an Activity***: During normal app use, the foreground activity is sometimes obstructed by other visual components that cause the activity to *pause*.As your activity enters the paused state, the system calls the onpause() method on your activity, which allows you to stop ongoing actions that should not continue while paused (such as a video) or persist any information that should be permanently saved in case the user continues to leave your app. If the user returns to your activity from the paused state, the system resumes it and callsthe onresume() method. indication that the user is leaving your activity.

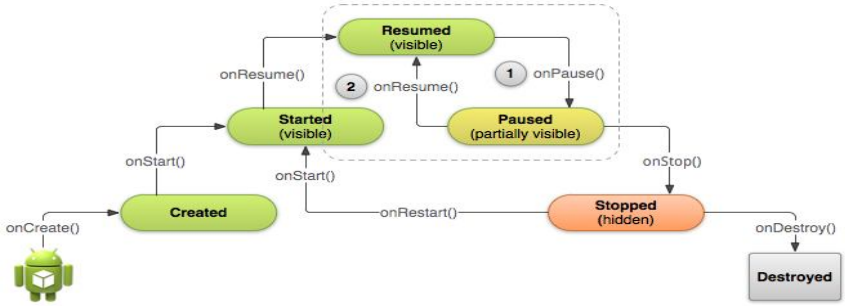

Figure 3: A sample diagram showing pausing and resuming an activity

3) ***3) Stopping and Restarting an Activity****:* Properly stopping and restarting your activity is an important process in the activity lifecycle that ensures your users perceive that your app is always alive and doesn't loose their progress. The Activity class provides two lifecycle methods, onStop() and onRestart(), which allow you to specifically handle how your activity handles being stopped and restarted. Unlike the paused state, which identifies a partial UI obstruction, the stopped state guarantees that the UI is no longer visible and the user's focus is in a separate activity (or an entirely separate app).

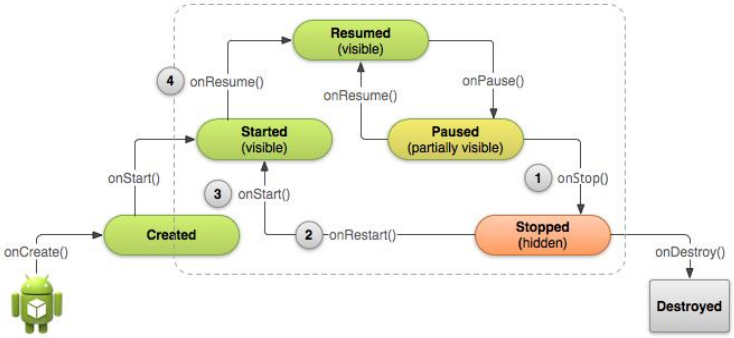

Figure 4: A sample diagram showing stopping and restarting an activity

4) ***Stop Your Activity****:* When your activity receives a call to the onStop() method, it's no longer visible and should release almost all resources that aren't needed while the user is not using it. Once your activity is stopped, the system might destroy the instance if it needs to recover system memory. In extreme cases, the system might simply kill your app process without calling the activity's final onDestroy() callback, so it's important you use onStop() to release resources that might leak memory.

5) ***Recreating an Activity***: There are a few scenarios in which your activity is destroyed due to normal app behavior, such as when the user presses the *Back* button or your activity signals its own destruction by calling finish(). The system may also destory your activity if it's currently stopped and hasn't been used in a long time or the foreground activity requires more resources so the system must shut down background processes to recover memory.

When your activity is destroyed because the user presses *Back* or the activity finishes itself, the system's concept of that Activity instance is gone forever because the behavior indicates the activity is no longer needed. However, if the system destroys the activity due to system constraints (rather than normal app behavior), then althought the actual Activity instance is gone, the system remembers that it existed such that if the user navigates back to it, the system creates a new instance of the activity using a set of saved data that describes the state of the activity when it was destroyed. The saved data that the system uses to restore the previous state is called the "instance state" and is a collection of key-value pairs stored in a Bundle object.

By default, the system uses the Bundle instance state to saves information about each View object in your activity layout (such as the text value entered into an EditText object). So, if your activity instance is destroyed and recreated, the state of the layout is automatically restored to its previous state. However, your activity might have more state information that you'd like to restore, such as member variables that track the user's progress in the activity.

In order for you to add additional data to the saved instance state for your activity, there's an additional callback method in the activity lifecycle that's not shown in the illustration from previous lessons. The method is onSaveInstanceState() and the system calls it when the user is leaving your activity. When the system calls this method, it passes the Bundle object that will be saved in the event that your activity is destroyed unexpectedly so you can add additional information to it. Then if the system must recreate the activity instance after it was destroyed, it passes the same Bundle object to your activity's onRestoreInstanceState() method and also to your onCreate() method.

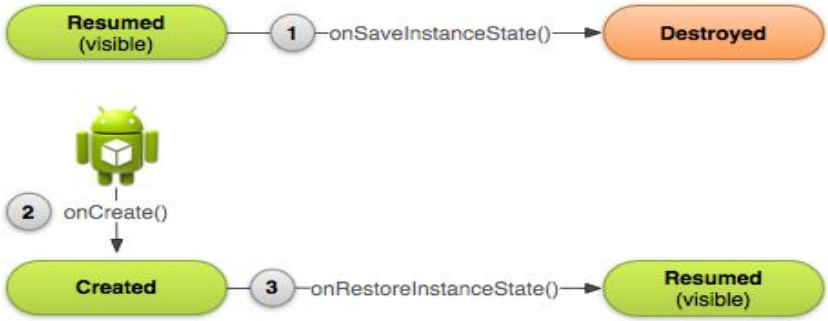

Figure 5: A sample diagram showing how to recreate an activity

EZY: Description of the Mobile Application

The objective is to make an all in one application for mobile phones which facilitates the following functionalities related to phonebook contacts:

Interface

Interface of the application has options of desired features of easy calling, easy messaging and easy emailing. All the contacts are retrieved into a separate dashboard and on toasting the contact image the set of options is displayed. This dashboard it

unique of its own as it has features to sort the list of contacts retrieved from the phonebook by name, recently contacted and most contacted.

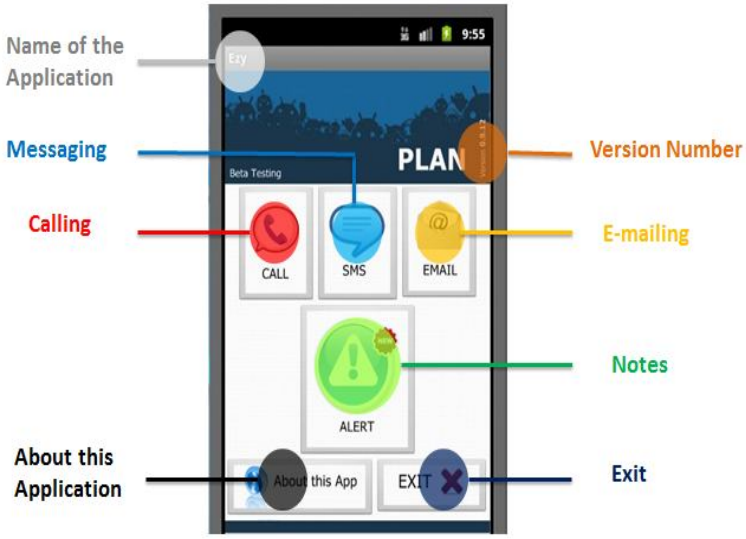

Figure 6: A sample diagram showing the interface of the EZY application.

Easy Calling

Easy calling allows planning a call which makes the user to specify the time beforehand at which the call should be initiated by the phone. Another option to make calling easy is the note flashing functionality. This functionality allows the user to save a note beforehand against a contact name and when that contact calls the user, the note flashes on the screen. Later when the user ends the call, it displays the option to keep the note or to suspend it. This way if the user chooses to keep the note, later if the same person calls the user again, the same note flashes until it is discarded. This feature makes the user to remember what he has to talk to the person by saving a note against him.

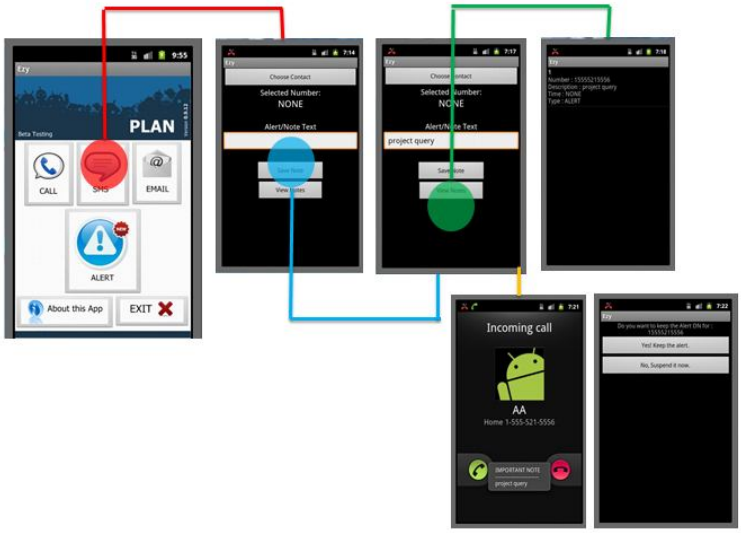

Figure 7: A sample diagram showing the easy calling functionality of the EZY application

The code for the implementation of a call is:

```
//Making a Call
   public void call(String phoneNumber) {
      try {
         Intent callIntent = new Intent(Intent.ACTION_CALL);
         callIntent.setData(Uri.parse("tel:"+ phoneNumber.trim()));
         startActivity(callIntent);
      } catch (ActivityNotFoundException activityException) {
          Log.e("Initiating Call", "Call failed", activityException);
      }
   }
```

The basic steps which we have followed to implement the note saving functionality are:

- Click event of the alert button
- Click event of the save button, once the user writes in the alert.
- Click event of the view button
- Functionality to flash the note when the concerned person calls the user.
- Functionality to provide the user an option to continue with or discard the existing note, when he hangs up the phone.

The code for making a note to flash when the person calls the user is:

```
protected void onActivityResult(int requestCode, int resultCode, Intent data)
 {
 super.onActivityResult(requestCode, resultCode, data);
if (requestCode == REQUEST_CONTACTPICKER)
                    {
                            if(resultCode == RESULT_OK)
                     {
Uri contentUri = data.getData();
                                    String contactId = contentUri.getLastPathSegment();
                                    Cursor cursor = getContentResolver().query(
                                          Phone.CONTENT_URI, null,
                                          Phone._ID + "=?",      // < - Note, not CONTACT_ID!
                                          new String[]{contactId}, null);
startManagingCursor(cursor);
                    Boolean numbersExist = cursor.moveToFirst();
                    int phoneNumberColumnIndex =
cursor.getColumnIndex(Phone.NUMBER);
                    String phoneNumber = "";
                    while (numbersExist)
                    {
                      phoneNumber = cursor.getString(phoneNumberColumnIndex);
                              phoneNumber = phoneNumber.trim();
                               phoneNumber = phoneNumber.replaceAll("[\\s\\-()]", "");
                      numbersExist = cursor.moveToNext();
                    }
stopManagingCursor(cursor);                         if(!phoneNumber.equals(""))
                                {
                                    Toast.makeText(getApplicationContext(), phoneNumber,
Toast.LENGTH_SHORT ).show();
```

```
                                TextView txtC =
(TextView)findViewById(R.id.textChosenNumber);
                                txtC.setText("" + phoneNumber);
```

Easy Messaging

Easy messaging allows planning a message which makes the user to specify the time beforehand at which the message should be sent by the phone.

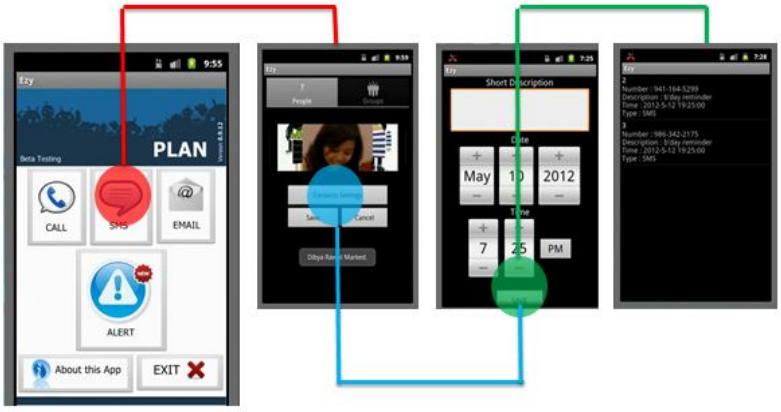

Figure 8: A sample diagram showing the easy messaging functionality of the EZY.

The code for the implementation of a planned message is:

```
//Sending an SMS
  public void sendSMS(String phoneNumber, String messageBody) {
     try {
        Intent sendIntent = new Intent(Intent.ACTION_VIEW);
        sendIntent.setData(Uri.parse("sms:" + phoneNumber));
        sendIntent.putExtra("sms_body", messageBody);

        startActivity(sendIntent);
     } catch (ActivityNotFoundException activityException) {
         Log.e("Initiatng SMS", "message failed", activityException);
     }
  }
```

Easy Mailing

Easy mailing allows planning a mail which makes the user to specify the time beforehand at which the mail should be sent by the phone.

The code for the implementation of a planned email is:

```
//Sending an Email
   public void sendEmail(String emailId, String subject, String messageBody) {
      try {
          Intent it = new Intent(Intent.ACTION_SEND);
          it.putExtra(Intent.EXTRA_EMAIL,  new String[]{ emailId.toString()});
          it.putExtra(Intent.EXTRA_TEXT, messageBody);
          it.setType("text/plain");
          it.putExtra(Intent.EXTRA_SUBJECT, subject);
          startActivity(Intent.createChooser(it, "Choose Email Client"));

           } catch (ActivityNotFoundException activityException){
Log.e("Initiatng Email", "Email Message failed", activityException);
      }
   }
```

Conclusions

The With the overall development of the project which took the complete one academic year we have achieved a successful implementation of the defined features in our ANDROID APP named “EZY” abbreviated form of “EASY” as it is a combined organizer.
As per our ideas ,this project will successfully integrate all the necessary applications in one with the use of latest technologies .This helps the user not to switch over from one application to another but to carry on all its work one a single application. The availability of such features into one is highly appreciable as it is still not available in the market. The easy to use interface of our application will help the user in easy handling of the application. Plan a message option will provide the user with an opportunity to save messages before hand and hence he will not miss on any important message that he is expected to send. The note flashing on the screen during an incoming call will help the user recollect as to what for the caller is calling him. All the features provided by our application will make the use a cell phone easy. The availability of such features into one is highly appreciable as it is still not available in the market.

The APP is made in android version 2.3.3 which is available as the Mobile Operating System for majority of the Android Phones e.g Samsung Galaxy, HTC and versions of Micromax and MTS.

References

[1] Google bets on Android future.
Web site http://news.bbc.co.uk/2/hi/technology/7266201.stm

[2] Android bug listing, website : http://code.google.com/p/android/issues/list

[3] Sdcard on enumerator tutorials.
Website: http://stackoverflow.com/questions/2506661/how-to-import-files-into-the-sdcard-on-emulator-in-android

[4] Android Developers: Website: http://www.androiddevelopers.com

[5] Virtual Box, Website:
http://developer.android.com/videos/index.html#v=Oq05KqjXTvs

[6] Windows Mobile: Web site: http://developer.windowsmobile.com