Biography Application

A

Project Report

submitted

in partial fulfillment

for the award of the Degree of

Bachelor of Technology

in Department of Computer Science & Engineering

(with specialization in Computer Science and Engineering)

Supervisor
Mrs. Priyanka Bhardwaj
Mr. Saurabh Ranjan Shrivastava

Mentor
Mrs. Anjana Sangwan

Submitted By
Archit Parnami
[10ESKCS023]

Department of Computer Science & Engineering

Swami Keshvanand Institute of Technology, Management & Gramothan, Jaipur

Rajasthan Technical University, Kota

May, 2014

CERTIFICATE

This is to certify that **Archit Parnami** students of B.Tech (Computer Science & Engineering) 8th semester has submitted his project report entitled “ Biography Application” under my guidance.

Mr. Saurabh Ranjan Shrivastava

Senior Lecturer

Department of Computer Science & Engineering

ACKNOWLEDGEMENT

We extend our sincere thanks to **Dr. C.M. Choudhary,** Head of the Computer Science & Engineering Department for providing me with the guidance and facilities for the Seminar.

We express our sincere gratitude to Seminar coordinator **Mr. Saurabh Ranjan Shrivstava** and **Mrs. Priyanka Bhardwaj** for their cooperation and guidance for preparing and presenting this seminar.

We also extend our sincere thanks to all other faculty members of Computer Science Department and my friends for their support and encouragement.

Archit Parnami

LIST OF FIGURES

CONTENTS

ABSTRACT

The idea behind this android application was to create a single application as template for other applications. So the final thought was to create a Biography Application of Famous Celebrities.
This Application would have a single Layout & all the content will be managed through a Microsoft Excel Spreadsheets. By having a single design & database connectivity with Excel one can create Multiple Android Apps.

Android has the largest installed base of any mobile OS and as of 2013, its devices alsosell more than Windows, iOS and Mac OS devices combined.As of July 2013 the Google Play store has had over 1 million Android apps published, and over 50 billion apps downloaded.A developer survey conducted in April–May 2013 found that 71% of mobile developers develop for Android.

CHAPTER 1

INTRODUCTION TO ANDROID

1.1 OVERVIEW

Android is an operating system based on the Linux kernel with a user interface based on direct manipulation, designed primarily for touchscreen mobile devices such as smartphones and tablet computers, using touch inputs, that loosely correspond to real-world actions, like swiping, tapping, pinching, and reverse pinching to manipulate on-screen objects, and a virtual keyboard. Despite being primarily designed for touchscreen input, it also has been used in televisions, games consoles, digital cameras, and other electronics.

As of 2011, Android has the largest installed base of any mobile OS and as of 2013, its devices alsosell more than Windows, iOS and Mac OS devices combined.As of July 2013 the Google Play store has had over 1 million Android apps published, and over 50 billion apps downloaded.A developer survey conducted in April–May 2013 found that 71% of mobile developers develop for Android.

Android's source code is released by Google under open source licenses, although most Android devices ultimately ship with a combination of open source and proprietary software. Initially developed by Android, Inc., which Google backed financially and later bought in 2005, Android was unveiled in 2007 along with the founding of the Open Handset Alliance—a consortium of hardware, software, and telecommunication companies devoted to advancing open standards for mobile devices.

Android is popular with technology companies which require a ready-made, low-cost and customizable operating system for high-tech devices. Android's open nature has encouraged a large community of developers and enthusiasts to use the open-source code as a foundation for community-driven projects, which add new features for advanced users or bring Android to devices which were officially, released running other operating systems. The operating system's success has made it a target for patent litigation as part of the so-called "Smartphone wars" between technology companies.

Android, Inc. was founded in Palo Alto, California in October 2003 by Andy Rubin (co-founder of Danger), Rich Miner (co-founder of Wildfire Communications, Inc.), Nick Sears (once VP at T-Mobile), and Chris White (headed design and interface development at WebTV) to develop, in Rubin's words "smarter mobile devices that are more aware of its owner's location and preferences". The early intentions of the company were to develop an advanced operating system for digital cameras, when it was realized that the market for the devices was not large enough, and diverted their efforts to producing a Smartphone operating system to rival those of

Symbian and Windows Mobile. Despite the past accomplishments of the founders and early employees, Android Inc. operated secretly, revealing only that it was working on software for mobile phones. That same year, Rubin ran out of money. Steve Perlman, a close friend of Rubin, brought him $10,000 in cash in an envelope and refused a stake in the company.

Google acquired Android Inc. on August 17, 2005; key employees of Android Inc., including Rubin, Miner, and White, stayed at the company after the acquisition. Not much was known about Android Inc. at the time, but many assumed that Google was planning to enter the mobile phone market with this move. At Google, the team led by Rubin developed a mobile device platform powered by the Linux kernel. Google marketed the platform to handset makers and carriers on the promise of providing a flexible, upgradable system. Google had lined up a series of hardware component and software partners and signaled to carriers that it was open to various degrees of cooperation on their part.

Speculation about Google's intention to enter the mobile communications market continued to build through December 2006. The unveiling of the iPhone, a touchscreen-based phone by Apple, on January 9, 2007 had a disruptive effect on the development of Android. At the time, a prototype device codenamed "Sooner" had a closer resemblance to a BlackBerry phone, with no touchscreen, and a physical, QWERTY keyboard. Work immediately began on re-engineering the OS and its prototypes to combine traits of their own designs with an overall experience designed to compete with the iPhone. In September 2007, *InformationWeek* covered an Evalueserve study reporting that Google had filed several patent applications in the area of mobile telephony.

On November 5, 2007, the Open Handset Alliance, a consortium of technology companies including Google, device manufacturers such as HTC, Sony and Samsung, wireless carriers such as Sprint Nextel and T-Mobile, and chipset makers such as Qualcomm and Texas Instruments, unveiled itself, with a goal to develop open standards for mobile devices.

1.2 ANDROID ARCHITECTURE

The figure below shows the Android's general architecture:

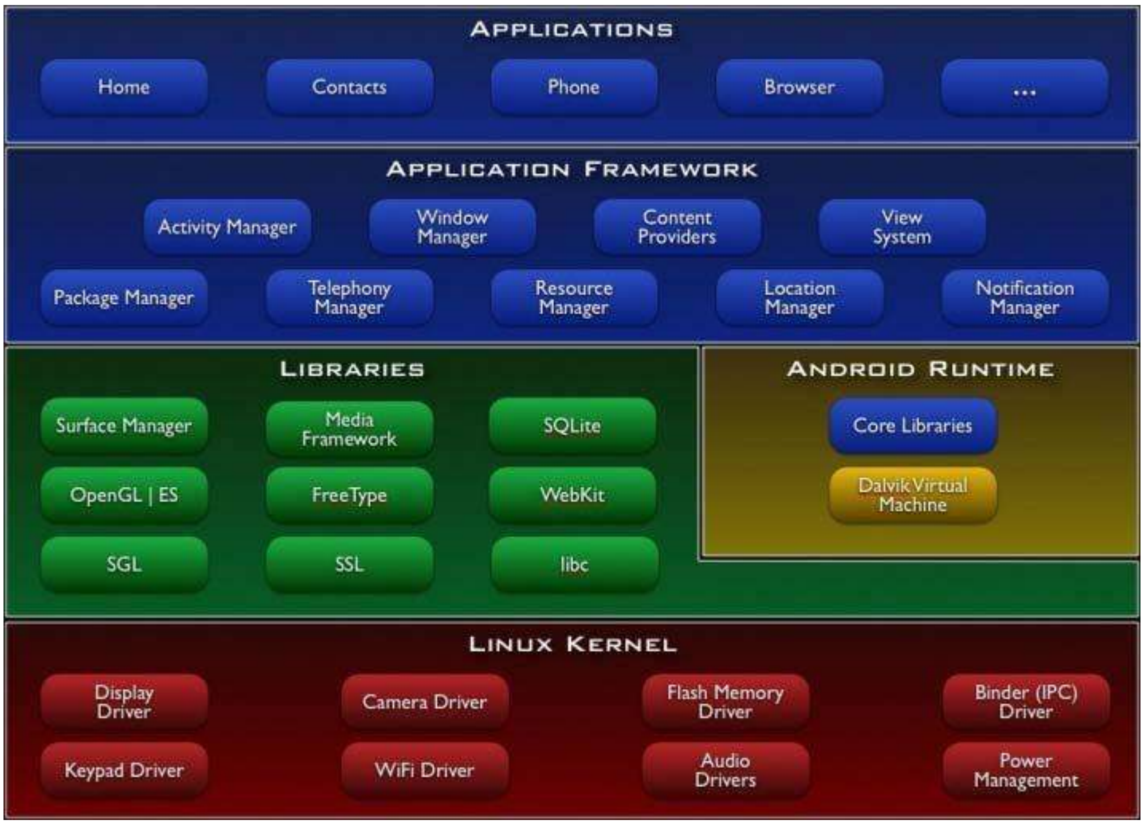


Figure 1.2: Android architecture

1. **Application layer:** Android provides a lot of applications whichcome with its release including an email client, SMS program, calendar, maps, browser, contacts, and others. All applications are written using the Java programming language. Moreover, the number of developers who interested in developing Android's application is increasing and provides a huge market of application to chose.

2. **Application framework layer:**enabling reuse and replacementof components. Not like Window mobile which restricts developer from system API, developers have full access to the same framework APIs used by the core's applications. The strength points of Android is that the application architecture is designed for reusing of components which means any application can publish its capabilities and any other application may the make use of those capabilities base on the definition of Intent which will be described later. Android provide a set of services and system, including:

i. **Views:** is a flexible definition. It can be a list, a grid, text box, button or even an embedded web browser. View in Android is very different from the definition of "view" in Symbian OS or Window mobile which often means the container in which graphic components are organized and displayed to user.

ii. **Content provider:**store and retrieve data and make it accessible to all applications. They're the only way to share data across applications; there's no common storage area that all Android packages can access.

iii. **Resource manager:** Resources are external files (that is, non-code files such as image, icon, and string for internationalization) that are used by developer's code and compiled into their application at build time. Android supports a number of different kinds of resource files, including XML, PNG, and JPEG files. The XML files have very different formats depending on what they describe. Resources are externalized from source code, and XML files are compiled into a binary, fast loading format for efficiency reasons. Strings likewise are compressed into a more efficient storage form.**Notification manager:** that enables all application to display custom alerts in the status bar notify user of what happen in the back ground.

iv. **Activity manager:** manages the lifecycle of application and provide common navigation back stack. An activity focuses on what user can do by interact with user. Activity will for example create a window and place UI component to be displayed to user. There are two important methods which are implemented by most subclass is:

OnCreate(Bundle) : where we initialize our activity, setup layout or get handle of each UI defined components.
OnPause(): where we deal with the event when user leaving our activity

3. **Libraries:** They are all written in C/C++ internally, but you'll becalling them through Java interfaces. These capabilities are exposed to developers through the Android application framework. Some of core libraries are:

i. **System C library** - a BSD-derived implementation of thestandard C system library (libc), tuned for embedded Linux-baseddevice**.**

ii. **Media Libraries** - based on PacketVideo'sOpenCORE;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

iii. **Surface Manager** - manages access to the displaysubsystem and seamlessly composites 2D and 3D graphic layers from multiple applications

iv. **LibWebCore**- a modern web browser engine whichpowers both the Android browser and an embeddable web view

v. **3D libraries** - an implementation based on OpenGL ES1.0 APIs; the libraries use either hardware 3D acceleration (where available) or the included, highly optimized 3D software rasterizer.

vi. **FreeType**- bitmap and vector font rendering

vii. **SQLite** - a powerful relational databaseengine available to all applications.

4. **Android Runtime:** A set of core libraries provides most thefunctionality available in the core library of java programming language. Android runtime includes the Dalvik Virtual Machine. Dalvik runs dex files, which are coverted at compile time from standard class and jar files. Dex files are more compact and efficient than class files, an important consideration for the limited memory and battery powered devices that Android targets. The core Java libraries are also part of the Android runtime. They are written in Java, as is everything above this layer. Here, Android provides a substantial set of the Java 5 Standard Edition packages, including Collections, I/O, and so forth.

5. **Dalvik Virtual Machine** which is optimized for mobile devices.Dalvik is a major piece of Google's Android, runs Java platform applications which have been converted into a compact DalvikExcutable format suitable for systems that are constrained in terms of memory and processor speed. Unlike most virtual machines an true java virtual machine which are stack machines, Dalvik VM is a register based architecture. Like the CISC vs. RISC debate, the relative merits of these two approaches is a subject of continuous argument but the underlying technology sometimes blurs the ideological boundaries. Moreover, the relative advantages of the two approaches depend on the interpretation/compilation strategy chosen. Generally, however, stack based machines must use instructions to load data on the stack and manipulate that data and thus require more instructions than register machines to implement the same high level code. However, the instructions in a register machine must encode the source and destination registers and therefore tend to be larger. This difference is primarily of importance to VM interpreters for whom opcode dispatch tends to be expensive and other factors are relevant for JIT compilation. Being optimized for low memory requirements, Dalvik VM use less space, has no JIT compiler and uses its own byte code, not java byte code.

6. **Linux Kernel:** Starting at the bottom is the Linux kernel. Androiduses it for its device drivers, memory management, process management and networking. However we will never be programming to this layer directly. Android relies on Linux kernel version 2.6 for core system services. The kernel also acts as an abstraction layer between the hardware and the rest of the software stack. One of the unique and powerful qualities of Android is that all applications have a level playing field which means that the applications Google writes have to go through the same public API that we use.

1.3 WHY ANDROID IS BETTER?

1. ***Applications***

Google applications

Android includes most of the time many Google applications like Gmail, YouTube or Maps. These applications are delivered with the machine most of the time, except in certain cases, such as some phones running android on which the provider has replaced Google applications by its own applications.

Widgets

With android, it is possible to use widgets which are small tools that can most often get information. These widgets are directly visible on the main window.

Android Market

This is an online software store to buy applications. Developers who created applications can add them into the store, and these applications can be downloaded by users, they can be both free and paid.

2. ***Multitasking***

Android allows multitasking in the sense that multiple applications can run simultaneously. With Task Manager it is possible view all running tasks and to switch from one to another easily.

3. ***SDK***

A development kit has been put at disposal of everybody. Accordingly, any developer can create his own applications, or change the android platform. This kit contains a set of libraries, powerful tools for debugging and development, a phone emulator, thorough documentation, FAQs and tutorials.

4. ***Modifiability***

This allows everyone to use, improve or transform the functions of Android for example transform the interface in function of uses, to transform the platform in a real system embedded Linux.

CHAPTER 2

ANDROID APPLICATION 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.

Once installed on a device, each Android application lives in its own security sandbox:

- The Android operating system is a multi-user Linux system in which each application is a different user.

- By default, the system assigns each application a unique Linux user ID (the ID is used only by the system and is unknown to the application). The system sets permissions for all the files in an application so that only the user ID assigned to that application can access them.Each process has its own virtual machine (VM), so an application's code runs in isolation from other applications.

- By default, every application runs in its own Linux process. Android starts the process when any of the application's components need to be executed, then shuts down the process when it's no longer needed or when the system must recover memory for other applications.

In this way, the Android system implements the *principle of least privilege*. That is, each application, by default, has access only to the components that it requires to do its work and no more. This creates a very secure environment in which an application cannot access parts of the system for which it is not given permission.

However, there are ways for an application to share data with other applications and for an application to access system services:

- It's possible to arrange for two applications to share the same Linux user ID, in which case they are able to access each other's files. To conserve system resources, applications with the same user ID can also arrange to run in the same Linux process and share the same VM (the applications must also be signed with the same certificate).
- An application can request permission to access device data such as the user's contacts, SMS messages, the mountable storage (SD card), camera, Bluetooth, and more. All application permissions must be granted by the user at install time.

That covers the basics regarding how an Android application exists within the system. The rest of this document introduces you to:

- The core framework components that define your application.
- The manifest file in which you declare components and required device features for your application.
- Resources that are separate from the application code and allow your application to gracefully optimize its behaviour for a variety of device configurations.

2.1 Application Components

Application components are the essential building blocks of an Android application. Each component is a different point through which the system can enter your application. Not all components are actual entry points for the user and some depend on each other, but each one exists as its own entity and plays a specific role—each one is a unique building block that helps define your application's overall behaviour.
There are four different types of application components. Each type serves a distinct purpose and has a distinct lifecycle that defines how the component is created and destroyed.

Here are the four types of application components:

(i) Activities

An *activity* represents a single screen with a user interface. For example, an email application might have one activity that shows a list of new emails, another activity to compose an email, and another activity for reading emails. Although the activities work together to form a cohesive user experience in the email application, each one is independent of the others. As such, a different application can start any one of these activities (if the email application allows it). For example, a camera application can start the activity in the email application that composes new mail, in order for the user to share a picture.
An activity is implemented as a subclass of Activity and you can learn more about it in the Activitiesdeveloper guide.

(ii) Services

A *service* is a component that runs in the background to perform long-running operations or to perform work for remote processes. A service does not provide a user interface. For example, a service might play music in the background while the user is in a different application, or it might fetch data over the network

without blocking user interaction with an activity. Another component, such as an activity, can start the service and let it run or bind to it in order to interact with it.
A service is implemented as a subclass of Service and you can learn more about it in the Servicesdeveloper guide.

(iii) Content providers

A *content provider* manages a shared set of application data. You can store the data in the file system, an SQLite database, on the web, or any other persistent storage location your application can access. Through the content provider, other applications can query or even modify the data (if the content provider allows it). For example, the Android system provides a content provider that manages the user's contact information. As such, any application with the proper permissions can query part of the content provider (such as ContactsContract.Data) to read and write information about a particular person.
Content providers are also useful for reading and writing data that is private to your application and not shared. For example, the Note Pad sample application uses a content provider to save notes.
A content provider is implemented as a subclass of ContentProvider and must implement a standard set of APIs that enable other applications to perform transactions. For more information, see the Content Providers developer guide.

(iv) Broadcast receivers

A *broadcast receiver* is a component that responds to system-wide broadcast announcements. Many broadcasts originate from the system—for example, a broadcast announcing that the screen has turned off, the battery is low, or a picture was captured. Applications can also initiate broadcasts—for example, to let other applications know that some data has been downloaded to the device and is available for them to use. Although broadcast receivers don't display a user interface, they may create a status bar notificationto alert the user when a broadcast event occurs. More commonly, though, a broadcast receiver is just a "gateway" to other components and is intended to do a very minimal amount of work. For instance, it might initiate a service to perform some work based on the event.
A broadcast receiver is implemented as a subclass of BroadcastReceiver and each broadcast is delivered as an Intent object. For more information, see the BroadcastReceiver class.
A unique aspect of the Android system design is that any application can start another application’s component. For example, if you want the user to capture a photo with the device camera, there's probably another application that does that and your application can use it, instead of developing an activity to capture a photo yourself. You don't need to incorporate or even link to the code from the camera application. Instead, you can simply start the activity in the camera application that captures a photo. When complete, the photo is even returned to your application so you can use it. To the user, it seems as if the camera is actually a part of your application.

When the system starts a component, it starts the process for that application (if it's not already running) and instantiates the classes needed for the component. For example, if your application starts the activity in the camera application that captures a photo, that activity runs in the process that belongs to the camera application, not in your application's process. Therefore, unlike applications on most other systems, Android applications don't have a single entry point (there's no main() function, for example).
Because the system runs each application in a separate process with file permissions that restrict access to other applications, your application cannot directly activate a component from another application. The Android system, however, can. So, to activate a component in another application, you must deliver a message to the system that specifies your *intent* to start a particular component. The system then activates the component for you.

2.2 The Manifest File

Before the Android system can start an application component, the system must know that the component exists by reading the application's AndroidManifest.xml file (the "manifest" file). Your application must declare all its components in this file, which must be at the root of the application project directory.
The manifest does a number of things in addition to declaring the application's components, such as:

- Identify any user permissions the application requires, such as Internet access or read-access to the user's contacts.
- Declare the minimum API Level required by the application, based on which APIs the application uses.
- Declare hardware and software features used or required by the application, such as a camera, bluetooth services, or a multitouch screen.
- API libraries the application needs to be linked against (other than the Android framework APIs), such as the Google Maps library.

AndroidManifest.xml is a powerful file in the Android platform that allows you to describe the functionality and requirements of your application to Android. However, working with it is not easy. Xamarin.Android attempts to fix this by allowing you to add custom attributes to your classes, which will then be used to automatically generate the manifest for you. Our goal is that 99% of people will never need to manually modify the manifest file.

2.3 Declaring Components

The primary task of the manifest is to inform the system about the application's components. For example, a manifest file can declare an activity as follows:

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

In the <application> element, the android:icon attribute points to resources for an icon that identifies the application.
In the <activity> element, the android:name attribute specifies the fully qualified class name of theActivity subclass and the android:label attributes specifies a string to use as the user-visible label for the activity.
You must declare all application components this way:

- <activity> elements for activities
- <service> elements for services
- <receiver> elements for broadcast receivers
- <provider> elements for content providers

Activities, services, and content providers that you include in your source but do not declare in the manifest are not visible to the system and, consequently, can never run. However, broadcast receivers can be either declared in the manifest or created dynamically in code (as BroadcastReceiver objects) and registered with the system by calling registerReceiver().
For more about how to structure the manifest file for your application, see The AndroidManifest.xml Filedocumentation.

2.4 Declaring Component Capabilities

As discussed above, in Activating Components, you can use an Intent to start activities, services, and broadcast receivers. You can do so by explicitly naming the target component (using the component class name) in the intent. However, the real power of intents lies in the concept of intent actions. With intent actions, you simply describe the type of action you want to perform (and optionally, the data upon which you'd like to perform the action) and allow the system to find a component on the device that can perform the action and start it. If there are multiple components that can perform the action described by the intent, then the user selects which one to use.

The way the system identifies the components that can respond to an intent is by comparing the intent received to the *intent filters* provided in the manifest file of other applications on the device.
When you declare a component in your application's manifest, you can optionally include intent filters that declare the capabilities of the component so it can respond to intents from other applications. You can declare an intent filter for your component by adding an <intent-filter> element as a child of the component's declaration element.
For example, an email application with an activity for composing a new email might declare an intent filter in its manifest entry to respond to "send" intents (in order to send email). An activity in your application can then create an intent with the “send” action (ACTION_SEND), which the system matches to the email application’s “send” activity and launches it when you invoke the intent with startActivity().

2.5 Declaring application requirements

There are a variety of devices powered by Android and not all of them provide the same features and capabilities. In order to prevent your application from being installed on devices that lack features needed by your application, it's important that you clearly define a profile for the types of devices your application supports by declaring device and software requirements in your manifest file. Most of these declarations are informational only and the system does not read them, but external services such as Google Play dread them in order to provide filtering for users when they search for applications from their device.
For example, if your application requires a camera and uses APIs introduced in Android 2.1 (API Level 7), you should declare these as requirements in your manifest file. That way, devices that do *not* have a camera and have an Android version *lower* than 2.1 cannot install your application from Google Play.
However, you can also declare that your application uses the camera, but does not *require* it. In that case, your application must perform a check at runtime to determine if the device has a camera and disable any features that use the camera if one is not available.
Here are some of the important device characteristics that you should consider as you design and develop your application:
Screen size and density
In order to categorize devices by their screen type, Android defines two characteristics for each device: screen size (the physical dimensions of the screen) and screen density (the physical density of the pixels on the screen, or dpi—dots per inch). To simplify all the different types of screen configurations, the Android system generalizes them into select groups that make them easier to target.
The screen sizes are: small, normal, large, and extra large.
The screen densities are: low density, medium density, high density, and extra high density.

By default, your application is compatible with all screen sizes and densities, because the Android system makes the appropriate adjustments to your UI layout and image resources. However, you should create specialized layouts for certain screen sizes and provide specialized images for certain densities, using alternative layout resources, and by declaring in your manifest exactly which screen sizes your application supports with the <supports-screens> element.
Many devices provide a different type of user input mechanism, such as a hardware keyboard, a trackball, or a five-way navigation pad. If your application requires a particular kind of input hardware, then you should declare it in your manifest with the <uses-configuration> element. However, it is rare that an application should require a certain input configuration.
Device features
There are many hardware and software features that may or may not exist on a given Android-powered device, such as a camera, a light sensor, bluetooth, a certain version of OpenGL, or the fidelity of the touchscreen. You should never assume that a certain feature is available on all Android-powered devices (other than the availability of the standard Android library), so you should declare any features used by your application with the <uses-feature> element.
Platform Version
Different Android-powered devices often run different versions of the Android platform, such as Android 1.6 or Android 2.3. Each successive version often includes additional APIs not available in the previous version. In order to indicate which set of APIs are available, each platform version specifies an API Level(for example, Android 1.0 is API Level 1 and Android 2.3 is API Level 9). If you use any APIs that were added to the platform after version 1.0, you should declare the minimum API Level in which those APIs were introduced using the <uses-sdk> element.
It's important that you declare all such requirements for your application, because, when you distribute your application on Google Play, the store uses these declarations to filter which applications are available on each device. As such, your application should be available only to devices that meet all your application requirements.

2.6 Application Resources

An Android application is composed of more than just code—it requires resources that are separate from the source code, such as images, audio files, and anything relating to the visual presentation of the application. For example, you should define animations, menus, styles, colors, and the layout of activity user interfaces with XML files. Using application resources makes it easy to update various characteristics of your application without modifying code and—by providing sets of alternative resources—enables you to optimize your application for a variety of device configurations (such as different languages and screen sizes).

For every resource that you include in your Android project, the SDK build tools define a unique integer ID, which you can use to reference the resource from your application code or from other resources defined in XML. For example, if your application contains an image file named logo.png (saved in the res/drawable/ directory), the SDK tools generate a resource ID named R.drawable.logo, which you can use to reference the image and insert it in your user interface.

One of the most important aspects of providing resources separate from your source code is the ability for you to provide alternative resources for different device configurations. For example, by defining UI strings in XML, you can translate the strings into other languages and save those strings in separate files. Then, based on a language *qualifier* that you append to the resource directory's name (such as res/values-fr/ for French string values) and the user's language setting, the Android system applies the appropriate language strings to your UI.

Android supports many different *qualifiers* for your alternative resources. The qualifier is a short string that you include in the name of your resource directories in order to define the device configuration for which those resources should be used. As another example, you should often create different layouts for your activities, depending on the device's screen orientation and size. For example, when the device screen is in portrait orientation (tall), you might want a layout with buttons to be vertical, but when the screen is in landscape orientation (wide), the buttons should be aligned horizontally. To change the layout depending on the orientation, you can define two different layouts and apply the appropriate qualifier to each layout's directory name. Then, the system automatically applies the appropriate layout depending on the current device orientation.

For more about the different kinds of resources you can include in your application and how to create alternative resources for various device configurations, see the Application Resources developer guide.

2.7 Process and Thread in Android

Android support multi-thread tasking allows developer to spawn additional threads for any process.

Process: The process where a component runs is controlled by the manifestfile. The component elements — <activity>, <service>, <receiver>, and <provider> — each have a process attribute that can specify a process where that component should run. These attributes can be set so that each component runs in its own process, or so that some components share a process while others do not. They can also be set so that components of different applications run in the same process — provided that the applications share the same Linux user ID and are signed by the same authorities. The <application> element also has a process attribute, for setting a default value that applies to all components.

All components are instantiated in the main thread of the specified process, and system calls to the component are dispatched from that thread. Separate threads are not created for each instance. Consequently, methods that respond to those calls — methods that report user actions and the lifecycle notifications always run in the main thread of the process. This means that no component should perform long or blocking operations (such as networking operations or computation loops) when called by the system, since this will block any other components also in the process. You can spawn separate threads for long operations.

Android may decide to shut down a process at some point, when memory is low and required by other processes that are more immediately serving the user. Application components running in the process are consequently destroyed. A process is restarted for those components when there's again work for them to do. When deciding which processes to terminate, Android weighs their relative importance to the user. For example, it more readily shuts down a process with activities that are no longer visible on screen than a process with visible activities. The decision whether to terminate a process, therefore, depends on the state of the components running in that process.

Thread: There will be somtime developers need to spawn a thread to dosome background work. Thread are created in code using standard Java Thread object. Android provides a number of convenience classes for managing threads like Looper for running a message loop within a thread, Handler for processing messages, and HandlerThread for setting up thread with a message loop. For example in My Delicious project, many time we need to perform network operation that takes long time to receive data. Solution is to perform that kind of work in a different thread while displaying a progress bar to user. Progress bar will be dimissed when sub-thread complete.

2.8 Activity lifecycle

Application components have a lifecycle — a beginning when Android instantiates them to respond to intents through to an end when the instances are destroyed. In between, they may sometimes be active or inactive,or, in the case of activities, visible to the user or invisible. Because Activity will be used most oftenly here we only discusses about the lifecycle of activities including the states that they can be in during their lifetimes, the methods that notify you of transitions between states, and the effect of those states on the possibility that the process hosting them might be terminated and the instances destroyed.

An activity has essentially three states:

- 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.
- It is *paused* if it has lost focus but is still visible to the user. 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.
- 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.

If an activity is paused or stopped, the system can drop it from memory either by asking it to finish , 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 methods:

+void onCreate(Bundle *savedInstanceState*) +void onStart()

+void onRestart() +void onResume() +void onPause()

+void onStop() +void onDestroy()

All activities must implement onCreate() to do the initial setup when the object is first instantiated. Many will also implement onPause() to commit data changes and otherwise prepare to stop interacting with the user.

Taken together, these seven methods define the entire lifecycle of an activity. There are three nested loops that you can monitor by implementing them:

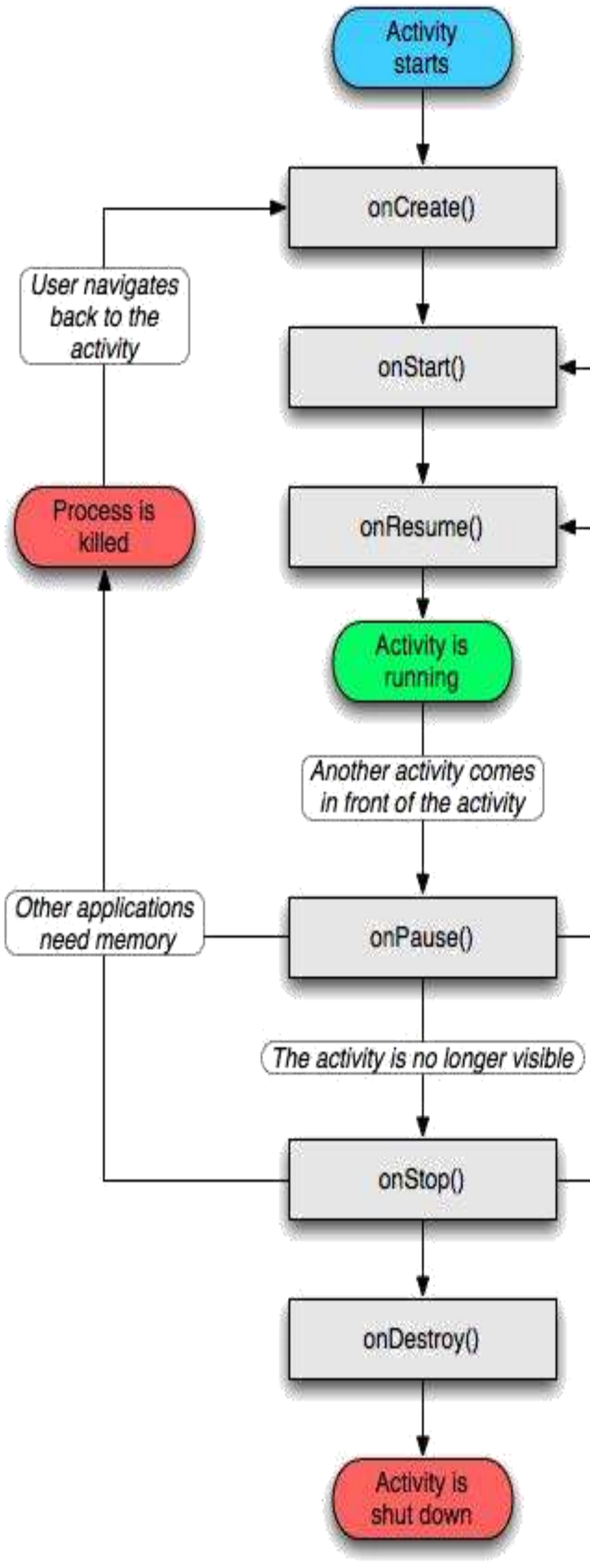
• The **entire lifetime** : An activity does all its initial setup of "global" state in onCreate(), and releases all remaining resources in onDestroy().

• The **visible lifetime** : During this time, the user can see the activity on-screen, though it may not be in the foreground and interacting with the user. Between these two methods, you can maintain resources that are needed to show the activity to the user. The onStart() and onStop() methods can be called multiple times, as the activity alternates between being visible and hidden to the user.

• The **foreground lifetime**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.

When the system, rather than the user, shuts down an activity to conserve memory, the user may expect to return to the activity and find it in its previous state. To capture that state before the activity is killed, you can implement an onSaveInstanceState() method for the activity. Android calls this method before making the activity vulnerable to being destroyed — that is, before onPause() is called. It passes the method a Bundle object where you can record the dynamic state of the activity as name-value pairs.

The following diagram illustrates these loops and the paths an activity may take between states. The colored ovals are major states the activity can be in. The square rectangles represent the callback methods you can implement to perform operations when the activity transitions between states.



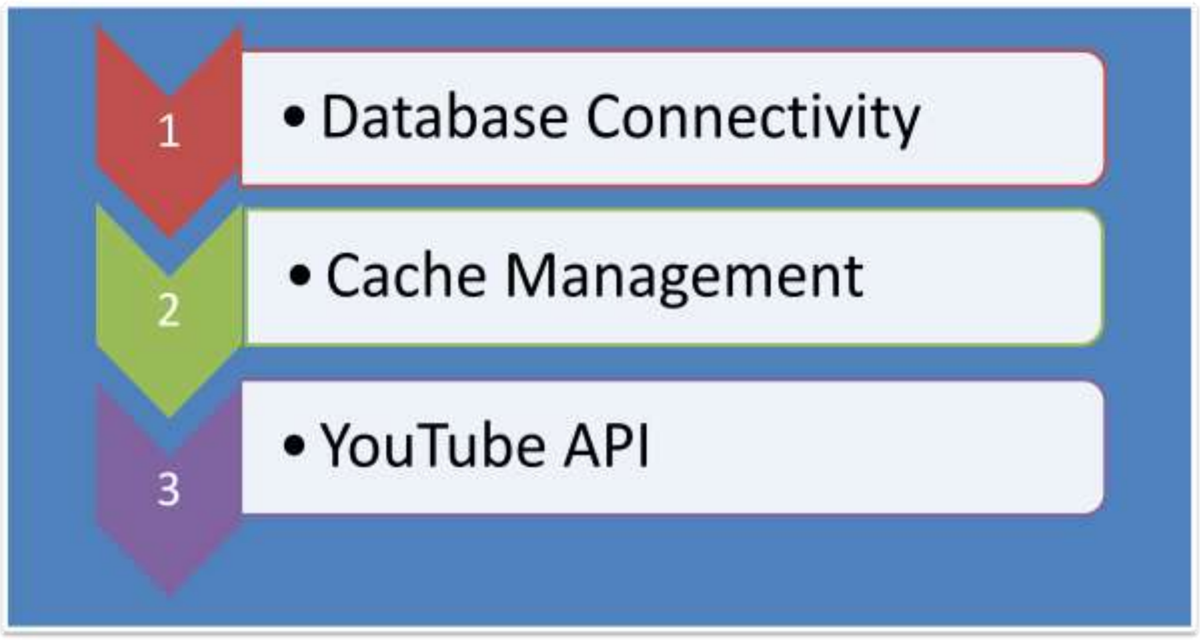
Activity starts
onCreate()
User navigates back to the activity
onStart()
onRestart()
Process is killed
onResume()
Activity is running
The activity comes to the foreground
Another activity comes in front of the activity
Other applications need memory
onPause()
The activity comes to the foreground
The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

Figure 2.8:Activity Lifecycle

CHAPTER 3

PROJECT DETAILS

3.1 GOAL



The Goal

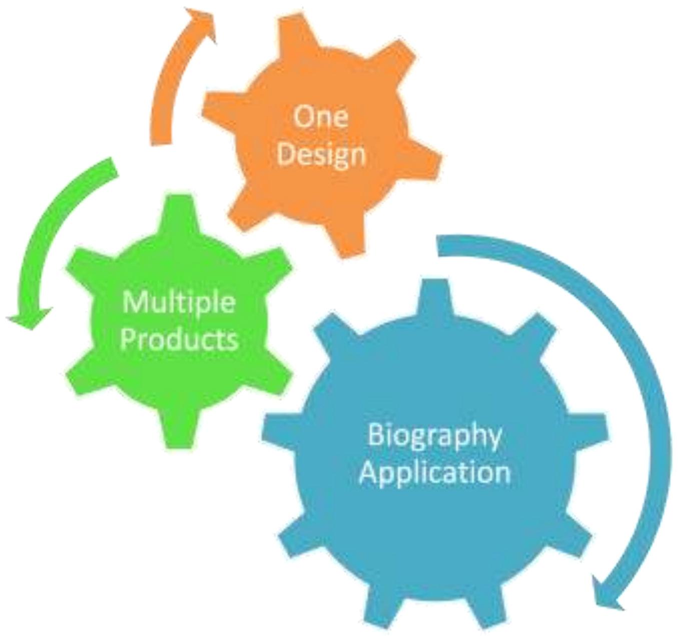
The Goal IN making this Android Application was to learn about the following:

1. Creating databases in Android Applications & connecting them with Microsoft Excel using JXL API.

2. Caching Images downloaded from the Internet For later use.

3. Playing YouTube Videos using YouTube API

3.2 Project Description



The Idea!

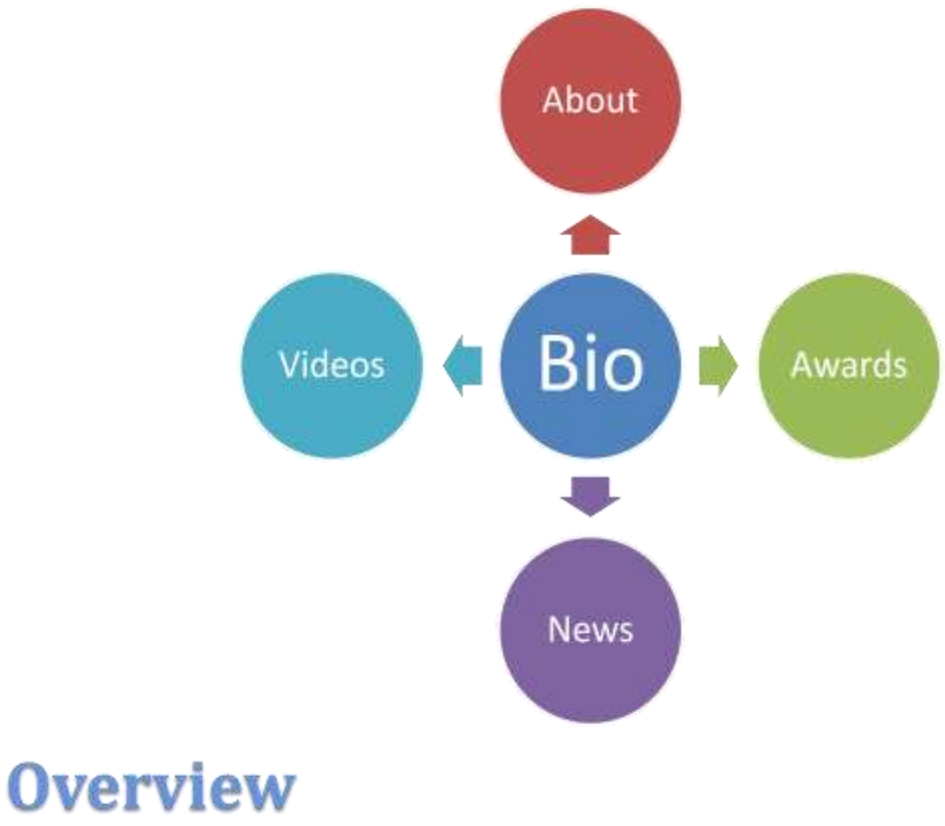
The idea behind this android application was to create a single application as template for other applications.
So the final thought was to create a Biography Application of Famous Celebrities.
This Application would have a single Layout & all the content will be managed through a Microsoft Excel Spreadsheets.
By having a single design & database connectivity with Excel one can create Multiple Android Apps.

3.3 The Sample Biography Application

- Develop a Biography Application of Gujarat’s Chief Minister Mr. Narendra Modi as a Sample Project.
- Use Excel Connectivity as a source of Information

The initial project setup was to create the Biography Application of Mr. NarendraModi as per the Project Requirement.
The Application has a splash Screen which displays an image of the celebrity while the application initializes its database.
This was achieved using Multi-Threading between the Splash screen display & Database initialization.
The Splash Screen lasts for few seconds generally depending upon the time taken by database to initialize which usually depends upon the processing power & RAM of the device running the application.

3.4 OVERVIEW

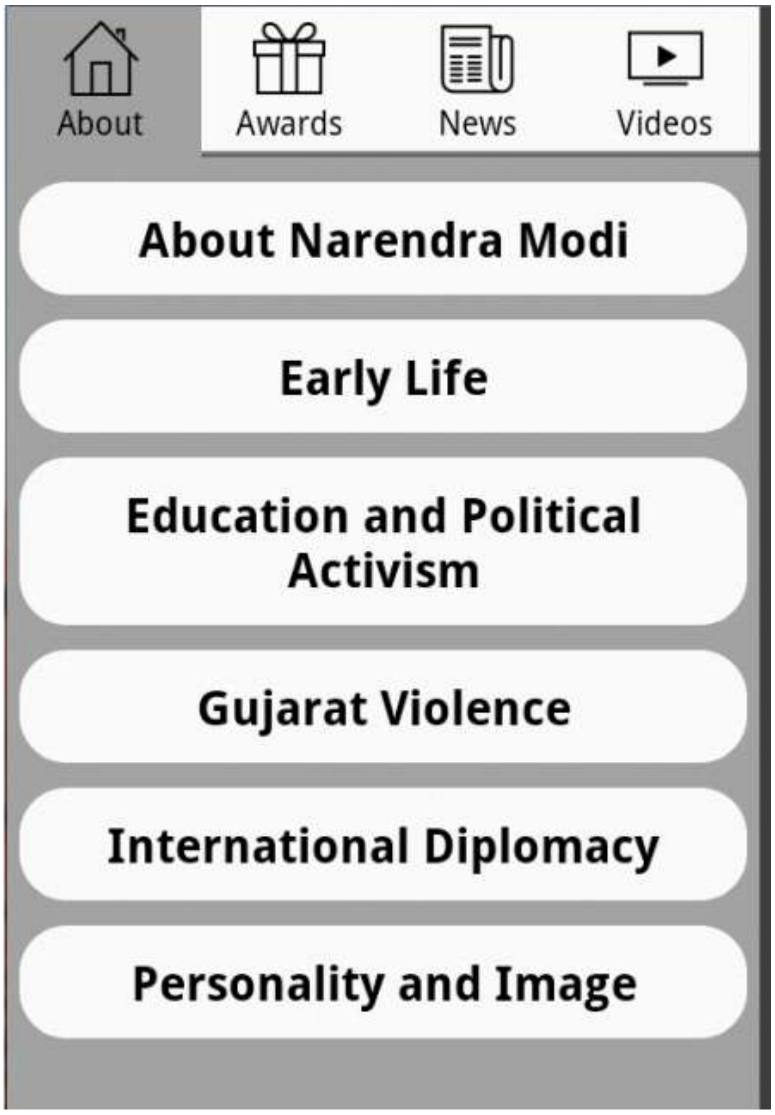


The Application has 4 different main layouts:

1. About
2. Awards
3. News
4. Videos

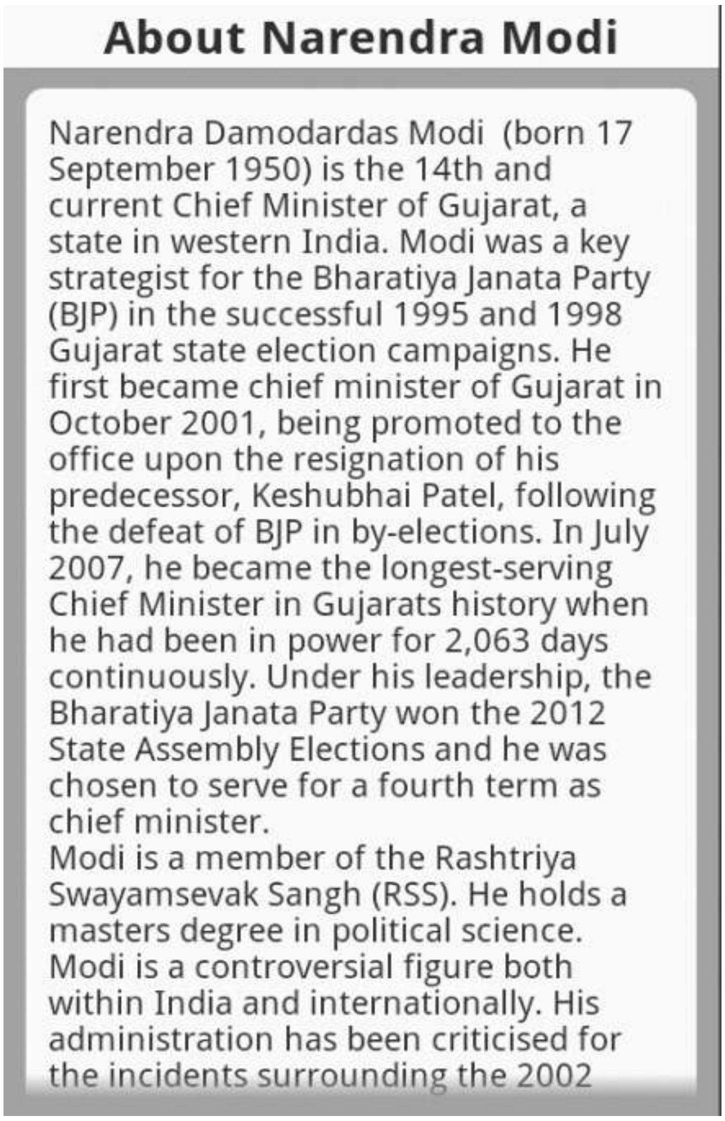
The content & Working of these layouts are discussed in the next section.

3.5 Interface



- This is the default tab & is displayed when the application first initializes.
- It uses an Android list to display contents.
- Each Item in the list opens in a Fragment.
- A Fragment is a reusable component of an application.
- Items are contained in a Scroll view.

3.6 Content View

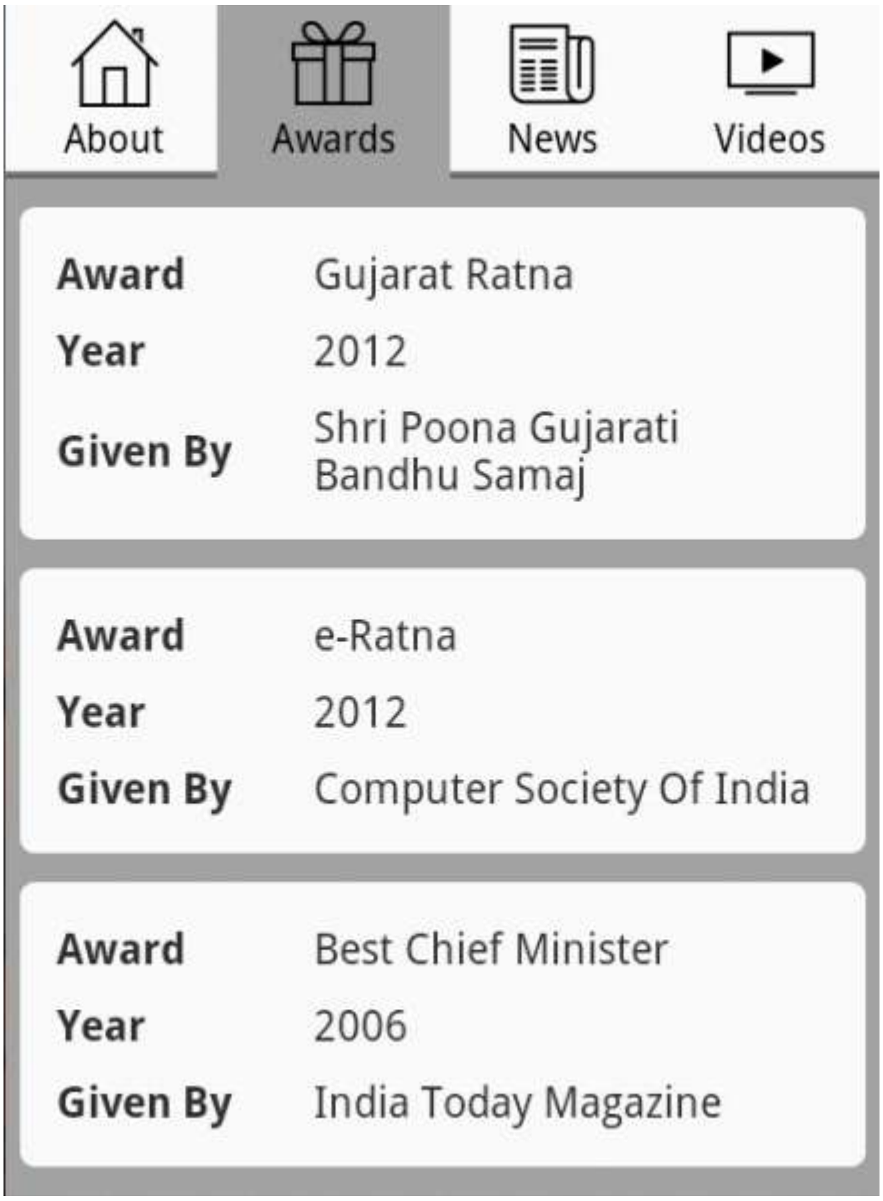


About Narendra Modi

Narendra Damodardas Modi (born 17 September 1950) is the 14th and current Chief Minister of Gujarat, a state in western India. Modi was a key strategist for the Bharatiya Janata Party (BJP) in the successful 1995 and 1998 Gujarat state election campaigns. He first became chief minister of Gujarat in October 2001, being promoted to the office upon the resignation of his predecessor, Keshubhai Patel, following the defeat of BJP in by-elections. In July 2007, he became the longest-serving Chief Minister in Gujarats history when he had been in power for 2,063 days continuously. Under his leadership, the Bharatiya Janata Party won the 2012 State Assembly Elections and he was chosen to serve for a fourth term as chief minister.
Modi is a member of the Rashtriya Swayamsevak Sangh (RSS). He holds a masters degree in political science.
Modi is a controversial figure both within India and internationally. His administration has been criticised for the incidents surrounding the 2002

- Contents of an Item of About Interface
- Contents are inside a Text View which itself is contained in a Scroll View.
- Swiping through this view opens the next element in the list.

3.7 Awards Interface



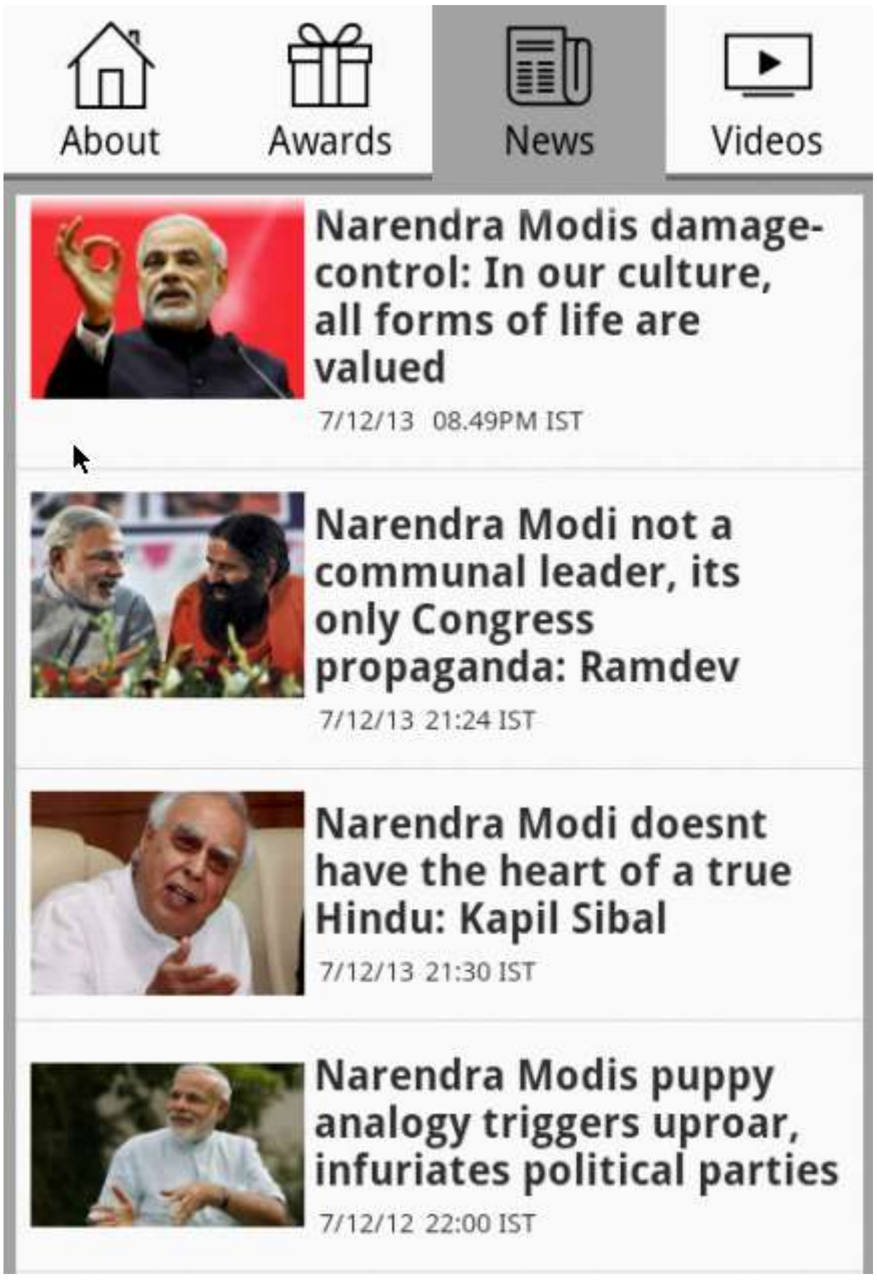
Award	Gujarat Ratna
Year	2012
Given By	Shri Poona Gujarati Bandhu Samaj

Award	e-Ratna
Year	2012
Given By	Computer Society Of India

Award	Best Chief Minister
Year	2006
Given By	India Today Magazine

- Award tab has a Linear Layout.
- Each Item in a award list is itself contained in a Linear Layout which contains three t views to display the three fields (Award, Year, Given By)

3.8 News Interface



Narendra Modis damage-control: In our culture, all forms of life are valued

7/12/13 08.49PM IST

Narendra Modi not a communal leader, its only Congress propaganda: Ramdev

7/12/13 21:24 IST

Narendra Modi doesnt have the heart of a true Hindu: Kapil Sibal

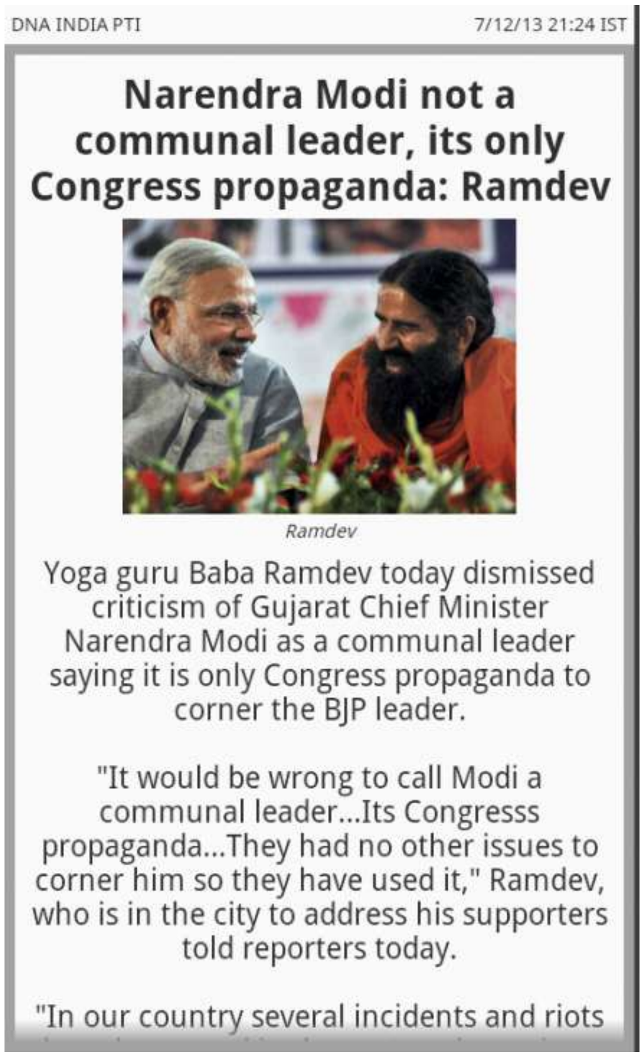
7/12/13 21:30 IST

Narendra Modis puppy analogy triggers uproar, infuriates political parties

7/12/12 22:00 IST

- The News Tan contains a list of news articles loaded from the Excel Worksheet at Application Initialization Step.
- Images are downloaded from the Internet & Cached to Memory & Disk.

3.9 News Article Interface



DNA INDIA PTI 7/12/13 21:24 IST

Narendra Modi not a communal leader, its only Congress propaganda: Ramdev

Ramdev

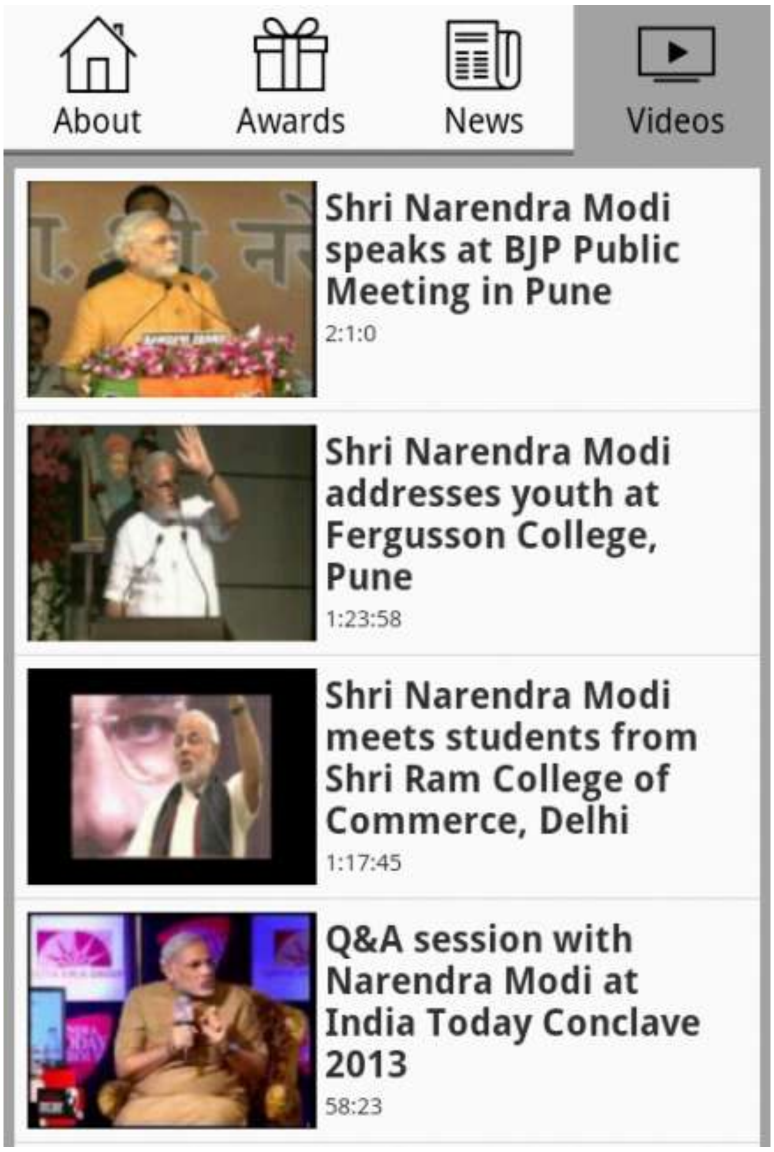
Yoga guru Baba Ramdev today dismissed criticism of Gujarat Chief Minister Narendra Modi as a communal leader saying it is only Congress propaganda to corner the BJP leader.

"It would be wrong to call Modi a communal leader...Its Congresss propaganda...They had no other issues to corner him so they have used it," Ramdev, who is in the city to address his supporters told reporters today.

"In our country several incidents and riots

- Each News article has a Source, Author, Date, Time, Title & a Body.
- All these fields are read from an Excel Worksheet only once & then saved Application’s internal Database.

3.10 Videos Interface



Shri Narendra Modi speaks at BJP Public Meeting in Pune
2:1:0

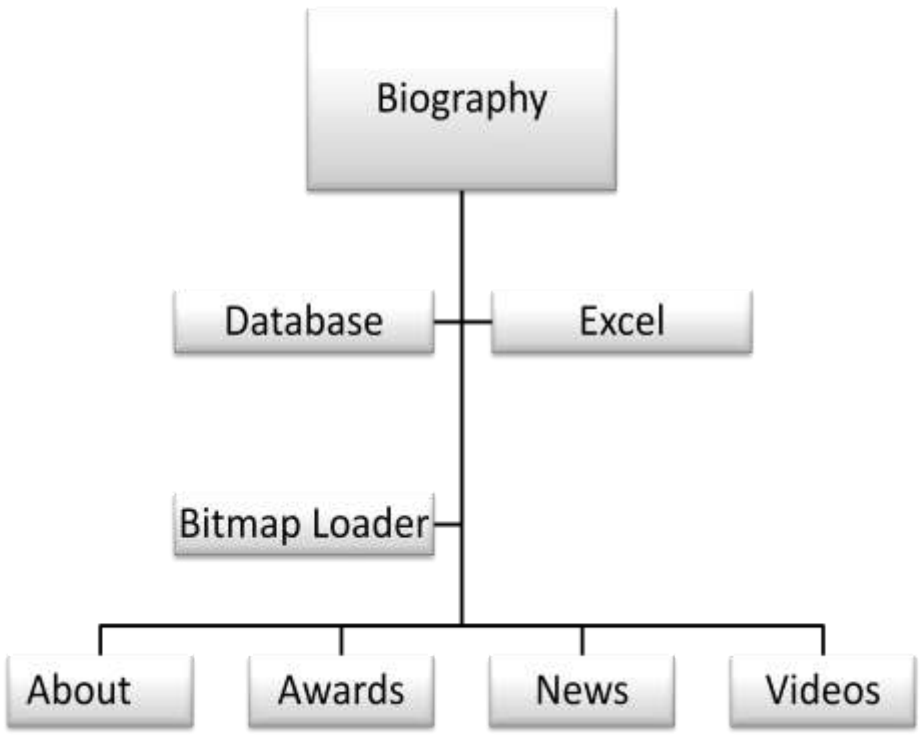
Shri Narendra Modi addresses youth at Fergusson College, Pune
1:23:58

Shri Narendra Modi meets students from Shri Ram College of Commerce, Delhi
1:17:45

Q&A session with Narendra Modi at India Today Conclave 2013
58:23

- The content of the video tab is not stored in the Excel Sheet Instead a link to the vi is kept.
- The link is parsed using the YouTube API which loads the Thumbnail & the Title.
- Thumbnails are then added to cache.
- Video is played using YouTube App.

3.11 Application Packages Used



Application Package API

The Application has the following packages:

1. App.Biography.Database
2. App.Biography.Excel
3. App.Biography.BitmapLoader
4. App.Biography.About
5. App.Biography.Awards
6. App.Biography.News
7. App.Biography.Videos

CHAPTER 4

TESTING

The Android testing framework, an integral part of the development environment, provides an architecture and powerful tools that help you test every aspect of your application at every level from unit to framework.

The testing framework has these key features:

- Android test suites are based on JUnit. You can use plain JUnit to test a class that doesn't call the Android API, or Android's JUnit extensions to test Android components. If you're new to Android testing, you can start with general-purpose test case classes such as AndroidTestCase and then go on to use more sophisticated classes.

- The Android JUnit extensions provide component-specific test case classes. These classes provide helper methods for creating mock objects and methods that help you control the lifecycle of a component.

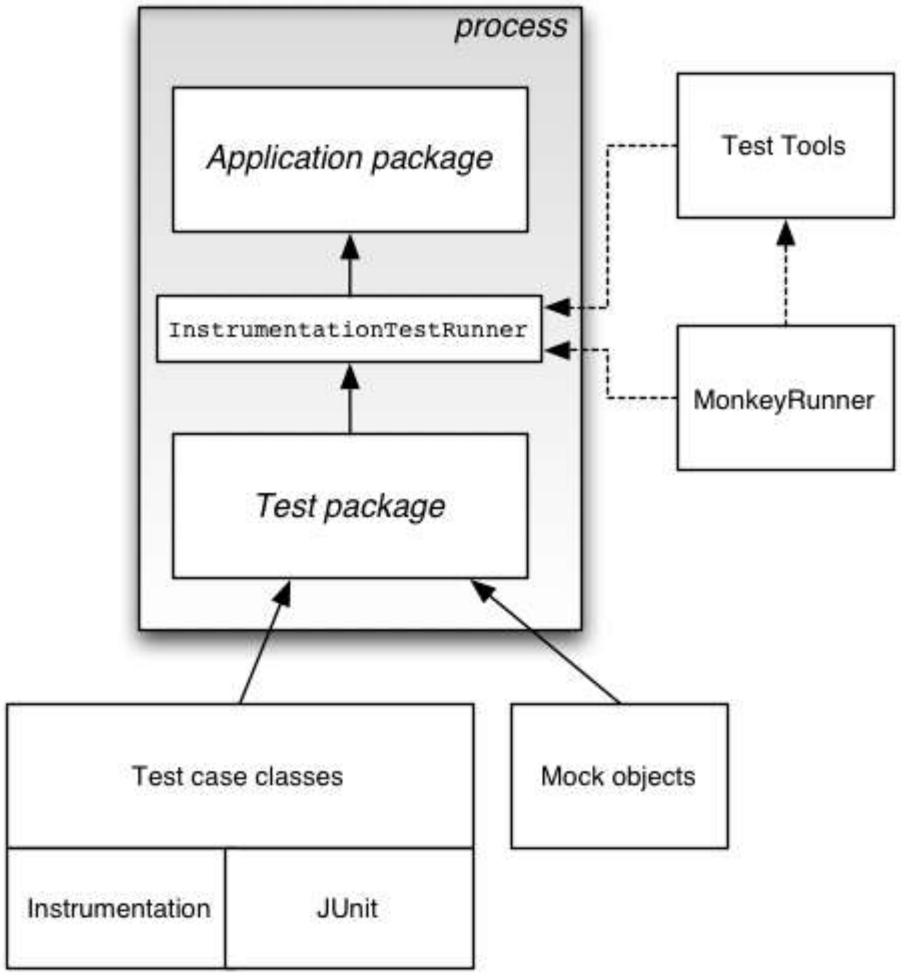
- Test suites are contained in test packages that are similar to main application packages, so you don't need to learn a new set of tools or techniques for designing and building tests.

- The SDK tools for building and tests are available in Eclipse with ADT, and also in command-line form for use with other IDEs. These tools get information from the project of the application under test and use this information to automatically create the build files, manifest file, and directory structure for the test package.

- The SDK also provides monkeyrunner, an API for testing devices with Python programs, and UI/Application Exerciser Monkey, a command-line tool for stress-testing UIs by sending pseudo-random events to a device.

- This document describes the fundamentals of the Android testing framework, including the structure of tests, the APIs that you use to develop tests, and the tools that you use to run tests and view results. The document assumes you have a basic knowledge of Android application programming and JUnit testing methodology.

The following diagram summarizes the testing framework:



4.1 Test Structure

Android's build and test tools assume that test projects are organized into a standard structure of tests, test case classes, test packages, and test projects.

Android testing is based on JUnit. In general, a JUnit test is a method whose statements test a part of the application under test. You organize test methods into classes called test cases (or test suites). Each test is an isolated test of an individual module in the application under test. Each class is a container for related test methods, although it often provides helper methods as well.

In JUnit, you build one or more test source files into a class file. Similarly, in Android you use the SDK's build tools to build one or more test source files into class files in an Android test package. In JUnit, you use a test runner to execute test classes. In Android, you use test tools to load the test package and the application under test, and the tools then execute an Android-specific test runner.

4.2 Test Projects

Tests, like Android applications, are organized into projects.A test project is a directory or Eclipse project in which you create the source code, manifest file, and other files for a test package. The Android SDK contains tools for Eclipse with ADT and for the command line that create and update test projects for you. The tools create the directories you use for source code and resources and the manifest file for the test package. The command-line tools also create the Ant build files you need.

You should always use Android tools to create a test project. Among other benefits, the tools:

Automatically set up your test package to use InstrumentationTestRunner as the test case runner. You must useInstrumentationTestRunner (or a subclass) to run JUnit tests.

Create an appropriate name for the test package. If the application under test has a package name ofcom.mydomain.myapp, then the Android tools set the test package name to com.mydomain.myapp.test. This helps you identify their relationship, while preventing conflicts within the system.

Automatically create the proper build files, manifest file, and directory structure for the test project. This helps you to build the test package without having to modify build files and sets up the linkage between your test package and the application under test. The

You can create a test project anywhere in your file system, but the best approach is to add the test project so that its root directory tests/ is at the same level as the src/ directory of the main application's project. This helps you find the tests associated with an application.

4.3 The Testing API

The Android testing API is based on the JUnit API and extended with a instrumentation framework and Android-specific testing classes.

JUnit

You can use the JUnit TestCase class to do unit testing on a class that doesn't call Android APIs. TestCase is also the base class for AndroidTestCase, which you can use to test Android-dependent objects. Besides providing the JUnit framework, AndroidTestCase offers Android-specific setup, teardown, and helper methods.

You use the JUnit Assert class to display test results. The assert methods compare values you expect from a test to the actual results and throw an exception if the comparison fails. Android also provides a class of assertions that extend the possible types of comparisons, and another class of assertions for testing the UI. These are described in more detail in the section Assertion classes

To learn more about JUnit, you can read the documentation on the junit.org home page. Note that the Android testing API supports JUnit 3 code style, but not JUnit 4. Also, you must use

Android's instrumented test runnerInstrumentationTestRunner to run your test case classes. This test runner is described in the section Running Tests.

Instrumentation

Android instrumentation is a set of control methods or "hooks" in the Android system. These hooks control an Android component independently of its normal lifecycle. They also control how Android loads applications.

Normally, an Android component runs in a lifecycle determined by the system. For example, an Activity object's lifecycle starts when the Activity is activated by an Intent. The object's onCreate() method is called, followed by onResume(). When the user starts another application, the onPause() method is called. If the Activity code calls the finish()method, the onDestroy() method is called. The Android framework API does not provide a way for your code to invoke these callback methods directly, but you can do so using instrumentation.

Also, the system runs all the components of an application into the same process. You can allow some components, such as content providers, to run in a separate process, but you can't force an application to run in the same process as another application that is already running.

With Android instrumentation, though, you can invoke callback methods in your test code. This allows you to run through the lifecycle of a component step by step, as if you were debugging the component.

The key method used here is getActivity(), which is a part of the instrumentation API. The Activity under test is not started until you call this method. You can set up the test fixture in advance, and then call this method to start the Activity.

Also, instrumentation can load both a test package and the application under test into the same process. Since the application components and their tests are in the same process, the tests can invoke methods in the components, and modify and examine fields in the components.

Test case classes

Android provides several test case classes that extend TestCase and Assert with Android-specific setup, teardown, and helper methods.

AndroidTestCase

A useful general test case class, especially if you are just starting out with Android testing, is AndroidTestCase. It extends both TestCase and Assert. It provides the JUnit-standard setUp() and tearDown() methods, as well as all of JUnit's Assert methods. In addition, it provides methods for testing permissions, and a method that guards against memory leaks by clearing out certain class references.

Component-specific test cases

A key feature of the Android testing framework is its component-specific test case classes. These address specific component testing needs with methods for fixture setup and teardown and component lifecycle control. They also provide methods for setting up mock objects. These classes are described in the component-specific testing topics:

Activity Testing

Content Provider Testing

Service Testing

Android does not provide a separate test case class for BroadcastReceiver. Instead, test a BroadcastReceiver by testing the component that sends it Intent objects, to verify that the BroadcastReceiver responds correctly.

ApplicationTestCase

You use the ApplicationTestCase test case class to test the setup and teardown of Application objects. These objects maintain the global state of information that applies to all the components in an application package. The test case can be useful in verifying that the <application> element in the manifest file is correctly set up. Note, however, that this test case does not allow you to control testing of the components within your application package.

InstrumentationTestCase

If you want to use instrumentation methods in a test case class, you must use InstrumentationTestCase or one of its subclasses. The Activity test cases extend this base class with other functionality that assists in Activity testing.

Assertion classes

Because Android test case classes extend JUnit, you can use assertion methods to display the results of tests. An assertion method compares an actual value returned by a test to an expected value, and throws an AssertionException if the comparison test fails. Using assertions is more convenient than doing logging, and provides better test performance.

Besides the JUnit Assert class methods, the testing API also provides the MoreAsserts and ViewAsserts classes:

MoreAsserts contains more powerful assertions such as assertContainsRegex(String, String), which does regular expression matching.

ViewAsserts contains useful assertions about Views. For example it containsassertHasScreenCoordinates(View, View, int, int) that tests if a View has a particular X and Y position on the visible screen. These assert simplify testing of geometry and alignment in the UI.

CONCLUSION

The initial project setup was to create the Biography Application of Mr. NarendraModi as per the Project Requirement. The Application has a splash Screen which displays an image of the celebrity while the application initializes its database. This was achieved using Multi-Threading between the Splash screen display & Database initialization.

The Splash Screen lasts for few seconds generally depending upon the time taken by database to initialize which usually depends upon the processing power & RAM of the device running the application. Contents of an item of about interface. Contents are inside a Text View which itself is contained in a Scroll View. Swiping through this view opens the next element in the list. Creating databases in Android Applications & connecting them with Microsoft Excel using JXL API. Caching Images downloaded from the Internet For later use. Playing YouTube Videos using YouTube API

Research Paper on Android

Android, Inc. was founded in Palo Alto, California, United States in October, 2003 by Andy Rubin (co-founder of Danger),Rich Miner (co-founder of Wildfire Communications, Inc.),Nick Sears (once VP at T-Mobile) and Chris White (headed design and interface development at WebTV).to develop, in Rubin's words "...smarter mobile devices that are more aware of its owner's location and preferences. Despite the obvious pastaccomplishments of the founders and early employees, Android Inc. operated secretively, admitting only that it was working on software for mobile phones.

Licensing

With the exception of brief update periods, Android has been available under a free software/open source license since 21 October 2008. Google published the entire source code (including network and telephony stacks) under an Apache License.Google also keeps the reviewed issues list publicly open for anyone to see and comment.

Even though the software is open-source, device manufacturers cannot use Google's Android trademark unless Google certifies that the device complies with their Compatibility Definition Document (CDD). Devices must also meet this definition to be eligible to license Google's closed-source applications, including Android Market.[40]

In September 2010, Skyhook Wireless filed a lawsuit against Google in which they alleged that Google had used the compatibility document to block Skyhook's mobile positioning service (XPS) from Motorola's Android mobile devices.In December 2010 a judge denied Skyhook's motion for preliminary injunction, saying that Google had not closed off the (maybe) possibility of accepting a revised version of Skyhook's XPS service, and that Motorola hadterminated their contract with Skyhook because Skyhook wanted to disable Google's location data collection functions on Motorola's devices, which would have violated Motorola's obligations to Google and its carriers.

The most recent released versions of Android are:

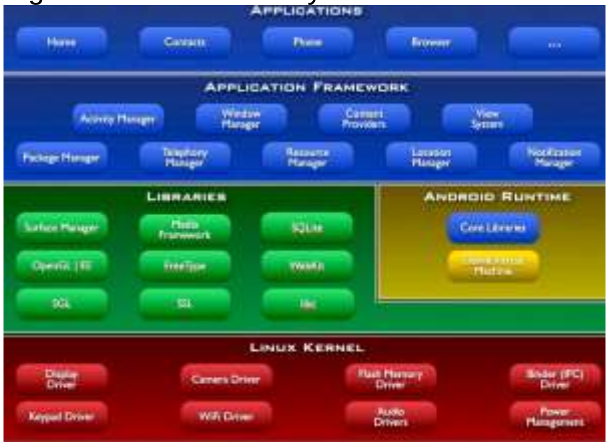
• **2.0/2.1 (Eclair)**, which revamped the user interface and introduced HTML5 and Exchange ActiveSync 2.5 support.

• **2.2 (Froyo)**, which introduced speed improvements with JIT optimization and the Chrome V8 JavaScript engine, and added Wi-Fi hotspot tethering and Adobe Flash support.

• **2.3 (Gingerbread)**, which refined the user interface, improved the soft keyboard and copy/paste features, and added support for Near Field Communication.

• **3.0 (Honeycomb)**, a tablet-orientedrelease which supports larger screen devices and introduces many new user interface features, and supports multicore processors and hardware acceleration for graphics. The Honeycomb SDK has been released and the first device featuring this version, the Motorola Xoom tablet, went on sale in February 2011.Google has chosen to withhold the development source code, which calls into question the "open-ness" of this Android release.Google claims this is done to eliminate manufacturers putting a Tablet-specific OS on phones, much like the previous autumn, where tablet manufacturers put a non-tabletoptimized phone OS (Android 2.x) on their Tablets resulting in bad user experiences.

Figure 2.1: Android system architecture



Software development kit

The Android software development kit (SDK) includes acomprehensive set of development tools.These include a debugger, libraries, a handset emulator (based on QEMU), documentation, sample code, and tutorials. The SDK is downloadable on the android developer website, or click here.Currently supported development platforms include computers running Linux (any modern desktop Linux distribution), Mac OS X 10.4.9 or later, Windows XP or later. The officially supported integrated development environment (IDE) is Eclipse (currently 3.5 or 3.6) using the Android Development Tools (ADT) Plugin, though developers may use any text editor to edit Java and XML files then use command line tools (Java
Development Kit and Apache Ant are required) to create, build and debug Android applications as well as control attached Android devices (e.g., triggering a reboot, installing software package(s) remotely).

A preview release of the Android SDK was released on 12 November 2007. On 15 July 2008, the Android Developer Challenge Team accidentally sent an email to all entrants in the Android Developer Challenge
announcing that a new release of the SDK was available in a "private" download area. The email was intended for winners of the first round of the Android Developer Challenge. The revelation that Google was supplying new SDK releases to some developers and not others (and keeping this arrangement private) led to widely reported frustration within the Android developer community at the time.

On 18 August 2008 the Android 0.9 SDK beta was released. This release provided an updated and extended API, improved development tools and an updated design for the home screen. Detailed instructions for upgrading are available to those already working with an earlier release.On 23 September 2008 the Android 1.0 SDK (Release 1) was released.According to the release notes, it included "mainly bug fixes, although some smaller features were added." It also included several API changes from the 0.9 version. Multiple versions have been released since.

Enhancements to Android's SDK go hand in hand with the overall Android platform development. The SDK also supports older versions of the Android platform in case developers wish to target their applications at older devices.

Development tools are downloadable components, so after one has downloaded the latest version and platform, older platforms and tools can also be downloaded for compatibility testing.

Google has also participated in the Android Market by offering several applications for its services. These applications include Google Voice for the Google Voice service, Sky Map for watching stars, Finance for their finance service, Maps Editor for their MyMaps service, Places Directory for their Local Search, Google Goggles that searches by image, Gesture Search for using finger-written letters and numbers to search the contents of the phone,Google Translate, Google Shopper, Listen for podcasts and My Tracks, a jogging application.

Linux compatibility

Android's kernel was derived from Linux but has been altered by Google outside the main Linux kernel tree.
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 GNU/Linux applications or libraries to Android. However, support for the X Window System is possible. Google no longer maintains the code they previously contributed to the Linux kernel as part of their Android effort, creating a separate version or fork of Linux. This was due to a disagreement about new features Google felt were necessary (some related to security of mobile applications).
The code which is no longer maintained was deleted in January 2010 from the Linux codebase.
Google announced in April 2010 that they will hire two employees to work with the Linux kernel community.However, as of January 2011, points of contention still exist between Google and the Linux kernel team: Google tried to push upstream some Android-specific power management code in 2009, which is still rejected today.

Conclusion

Mobile device processing capabilities have increased remarkably through the latest years. This makes it possible to build more complex applications targeted for mobile devices. This study and its results, shows how architectures and systems mostly designed for desktop usage like Web service invocation with SOAP messaging, now also is possible to be used on mobile platforms like Android.
With the help from faster and more available mobile networks, accessingWeb services directly with SOAP messaging is definitely possible on Android. However, the high network availability on such mobiledevices,
also makes an architectural alternative like a HTML frontendincreasingly competitive. This is strengthen by the increasing number of mobile platforms application developers must support and by observing the trend on desktop computers where web-based applications like Google Docs1 have become a strong alternative to native desktop applications.
It would be useful to test the developed proof of concept application on an actual Android device such as the HTC Dream. The same benchmarks described in this thesis, are repeatable and the results can be compared.
Such a comparison enables the architectural choices and Web service design practices to be revised based upon more accurate results.
Further, a study on how SOAP invocations' affect battery lifetime will be valuable in order to evaluate how applications can be used in real world situations. During this study, a set of guidelines for accessing Web services directly from Android was proposed in section. An analysis on how the MPower platform supports these guidelines might help to identify how available Web services can be changed in order to better support direct invocation from mobile devices. Such a study might also be applicable to other SOA platforms, making it a valuable research contribution.

References

- www.ieee.org.
- www.webcitation.org.
- www.w3.org.
- www.developer.android.com.
- www.source.android.com.

REFERENCES

- www.source.android.com.

- www.ieee.org.
- www.webcitation.org.
- www.w3.org.
- www.developer.android.com.
- www.source.android.com.