

Helsinki Metropolia University of Applied Sciences
Degree Programme in Information Technology

Fidelis Nwobuwa Ololube

**Development of Reporter's Video Recorder Based on
Android**

Master of Engineering Thesis, 29 September 2010

Supervisor: Kari Salo, Principal Lecturer

Language Advisor: Taru Sotavalta, Senior Lecturer

Author	Fidelis Nwobuwa Ololube
Title	Development of Reporter's Video Recorder Based on Android
Number of Pages	74
Date	29 September 2010
Degree Programme	Information Technology
Degree	Master of Engineering
Supervisor	Kari Salo, Principal Lecturer
<p>The main goal of this project was to develop an easy to use Android-based video recorder that would be capable of recording events, collecting the required meta-data automatically and uploading the file to the server with the availability of WLAN. This project was carried out for Sanoma Oy as part of the Next Media research program.</p> <p>In this project, the mobile side and the server side was developed. The Android SDK incorporated with Eclipse IDE was used to develop the mobile side while PHP was used for the server side. The application also made use of some Android-specific APIs such as the location-based API, the media API, and the network API. The project serves as a prototype. The first version has been translated into the Finnish language and published in Android Market on the 6 June 2010, with the name "Hyperlocal Videointi".</p> <p>The Application was demonstrated to the Sanoma Oy staff, and a modified version of the application was evaluated by researchers from the Tampere University of Technology, the unit of Human-Centered Technology, and Managing Editor from Sanoma Kaupunkilehdet. Positive feedback was given as a result of the evaluation. The project creates easier possibilities for future developers who might show an interest in this project.</p>	
Keywords	Android, Video View, Intent, Broadcast Receiver, PHP, ADT, media recorder, location-based, upload, download.

Acknowledgements

This project was carried out as a partial fulfillment of the requirements in achieving the Master of Engineering degree in Information Technology. The project was developed as part of the TEKES sponsored NextMedia Research Program.

I would like to thank all those who have helped me morally and otherwise in making this project a success. I especially want to thank my supervisor, Mr. Kari Salo, for his surjection and guidance in this project work, and to Mr. Matti Peltoniemi, for providing and configuring the school's server which was used for the server-side development of the application.

My deepest gratitude goes to my family for their encouragement, love and support towards me, especially to my father, H.R.H Eze Dr. A.O Ololube (JP) for his love towards all his children and his ever-ready attitude towards giving good education to his children. From my mother, H.R.H Mrs Comfort Ololube, I simply cannot ask for more, I have no words to express the love she has for her children.

CONTENTS

Abstract

Abbreviations.....	6
1 Introduction.....	8
2 Co-creations	9
2.1 Overview	9
2.2 The Foundation of co-creation	9
2.3 Co-creation Business Models.....	10
2.4 User-generated Content	11
2.5 Citizen Journalism	13
3 Application Development Environments.....	15
3.1 Overview	15
3.2 Installations and Plug-in	16
3.3 Android Architecture	17
3.4 Android Application Life Cycle.....	19
3.5 Android Application Framework	21
3.5.1 Activity Manager	21
3.5.2 Content Provider	23
3.5.3 Resource Manager	23
3.5.4 Location Manager.....	24
3.5.5 Notification Manager	25
4 Media API and Networks	26
4.1 Overview	26
4.2 Using the Media API	26
4.3 Media Player	27
4.4 Media Recorder	28
4.5 WLAN Connectivity.....	30
4.5.1 Broadcast Receiver	31
4.5.2 Background Services	31
4.5.3 Communication Protocols	32
4.5.4 Requirements to Port Files into YouTube.....	33
5 Reporter's Video Recorder Based on Android	37
5.1 Overview	37
5.2 Requirements.....	37
5.3 Design	38
5.4 Implementation.....	42
5.4.1 Application Menu	42
5.4.2 Video Recorder.....	44
5.4.3 Audio Recorder.....	46

5.4.4 Video Player	48
5.4.5 Audio Player	50
5.4.6 Sending file to the server.....	51
5.4.7 Search for WLAN	58
5.4.8 Deleting file	59
5.4.9 Application Manifest File.....	60
5.5 Feedback and Future Implementations	62
6 Conclusion	63
References	65
Appendices	67
Appendix A: Application main view.....	67
Appendix B: Video recording	69
Appendix C: Uploading files	71
Appendix D: Downloading video files	73
Appendix E: Playing Video	74

Abbreviations

3G	Third Generation
3GPP	3 rd Generation Partnership Project
ADT	Android Development Tool
API	Application Programming Interface
BBC	British Broadcasting Corporation
CBS	Columbia Broadcasting System
CGM	Consumer generated media
EDGE	Enhanced Data rates for Global Evolution
GPRS	General packet radio service
GPX	GPX Exchange Format
HTTP	Hypertext Transfer Protocol
IDE	Integrated development environment
IP	Internet Protocol
JAR	Java Archive
JDK	Java Development Kit
JPEG	Joint Photographic Experts Group
KML	Keyhole Markup Language
LRU	Least recently used
MP3	MPEG-1 Standard-based lossy audio compression method
MPEG4	Motion Picture Expert Group

OECD	Organization for Economic Co-operation and Development
OGG	An open standard multimedia container file format
OS X	Mac OS X version of operating system
PHP	Hypertext Preprocessor
PNG	Portable Network Graphics
SDcard	Secure Digital Card
SDK	Software Development Kit
SMS	Short Message Service
SMTP	Simple Mail Transfer Protocol
TCP	Transmission Control Protocol
UCC	User created content
UGC	User generated content
UI	User Interface
UMG	Universal Music Group
URL	Uniform Resource Locator
VM	Virtual Machine
Wi-Fi	A trademark of the Wi-Fi Alliance
WLAN	Wirless Local Area Network
XML	Extensible Markup Language
XMPP	Extensible Messaging and Presence Protocol

1 Introduction

Mobile phones are among the most used electronic devices due to their portability and assistance to mankind. The importance is not only for the use of calling loved ones and colleagues but also for the numerous applications that can be installed in it. The possibility of creating a quality video is one of the most interesting applications that a modern hand-set has to offer.

Content generation becomes more interesting when video clips are included. Communicating with the audience on-line has become of great interest to establishments around the world. This project aims to provide a better way of making work easier for reporters, citizen media, freelancers and many other business sectors to provide and share more detailed information with their clients. Co-creation opens a new collaborative ideology where establishments do not only have to imagine the needs of their clients but also involve them in the development process. The possibility of effectively handling user-generated content is one of the strengths of this type of work.

The Android mobile platform provides a media framework that helps developers customize and add greater efficiency to their application. Android is a software stack for mobile devices that run on top of Linux Kernel. Android comprises an operating system, middleware and key applications. It was first developed by Android Inc, which was later bought by Google and then by the Open Handset Alliance. The location-based application program interface (API), Notification API, Networking API and other APIs are used in this project.

The goal of this project is to develop an easy-to-use Android-based video recorder that would be capable of collecting the required meta-data automatically and upload the data to a server with the availability of WLAN.

2 Co-creations

2.1 Overview

Co-creation is a form of market or business strategy that focuses on the generation and ongoing realization of mutual firm-customer value. It views markets as forums for firms and active customers to share, combine and renew each other's resources and capabilities to create value through new forms of interaction, services and learning mechanisms. This makes a clear difference from the traditional active firm-passive consumer market construct of the past. Values from co-creation experiences are projected in the form of personalized, unique experiences for the customer (value-in-use) and ongoing revenue, learning and enhanced market performance drivers for the firm (loyalty, relationships, customer word of mouth).

Value is co-created with customers if and when a customer is able to personalize his or her experience using a firm's product-service proposition to a level that is best suited to get his or her job(s) or tasks done and which allows the firm to derive greater value from its product-service investment in the form of new knowledge, higher revenues/profitability and/or superior brand value/loyalty [1]. An example of co-creation includes Lego, Linux, Skype, Wikipedia, and E-bay.

2.2 The Foundation of co-creation

The key building blocks of co-creation for business have been identified to be the following: dialogue, access, risk assessment and transparency [2, 5; 3, 4].

- **Dialogue** - facilitates knowledge sharing and a shared level of understanding between the parties involved. The focus of the customer is not only on the ownership of the product but also on the access to desirable experiences.
- **Risk assessment** - is a critical step in all co-creation activities to ensure that client input is being encouraged and utilized appropriately. Since companies and consumers are involved in the co-creation of values, consumers demand more information about the potential risks of goods and services, which also implies that they bear more responsibilities of dealing with those risks.

- **Transparency** – Clients need reassurance that their input and ideas will be considered equitably when materials and systems are being planned and designed. The fact that a company's information is readily accessible to the consumer makes it necessary to create trust between the two parties.

In view of initiating a co-creative experience to engage a site visitor or client, some basic principles have to be adhered to for efficiency and clarity. These include the following:

- Value and acknowledge a site visitor's input.
- Provide easy, accessible means to communicate ensuring that he/she know where to respond without glitches.
- Clearly explain what is needed from he/she, ensure that it is easy for them to answer the questions or give input on an idea.
- Allow all clients to give input; the next great idea might eventually come from there.
- Prove to be listening; keep them up to date on how their input is being applied.
- Make the process enjoyable and meaningful.
- Acknowledge their input publicly; let others know how helpful they have been.
- Keep them engaged; respond to them, both privately and publicly [2, 6].

2.3 Co-creation Business Models

Business models are the way one earns revenue from customers and they are defined by the following principles: target market, value proposition, position in value chain and network, revenue model and cost structure, and competitive strategy. Putting the business model into the following components will allow one to analyze how co-creation will fit the model.

- **Target Market**

Any market can be a target. Some markets are better fit for co-creation firms, such as markets with easily modifiable products. Many software manufacturers fall into this category, especially those that serve software as a service. Google and Salesforce are examples of companies that can rapidly develop and change software depending on customer requirements and satisfaction.

- Value proposition

The value proposition involves the firm providing the co-creation experience to the consumer. Here, the customer connection, customer understanding or customer base can be a strong value proposition for co-creation firms.

- Position in value chain and network

Co-creation firms can be anywhere in the value chain. This is especially true if the company has developed a distinct relationship with the customer. This issue must be different from conventional firms.

- Revenue model and cost structure

This is the most significant factor for co-creation. Because several different participants literally co-create value at co-creation business models, it is not easy to calculate the portion of a contribution from each participant and to distribute value or revenue to each. This will be examined within the administrative and software developer arm of an organization.

- Competitive strategy

Co-creation firms have tight relationships with customer groups through its co-creation experience. This relationship positively affects switching costs and lock-in of the customer. Also, co-creation business tends to have strong network effects [3, 5-6].

2.4 User-generated Content

User-generated content (UGC), also known as consumer-generated media (CGM) or user-created content (UCC), refers to various types of media content, accessible by the public, which is produced by end-users. This term entered mainstream usage during 2005 having arisen in web publishing and new media content production circles. It is used for a wide range of applications including problem processing, news, gossip and research, reflecting the expansion of media production through new technologies that are accessible and affordable to the general public. All digital media technologies are

included, such as question-answer databases, digital video, blogging, pod casting, mobile phone photography and wikis. In addition to these technologies, user generated content may also employ a combination of open source, free software, and flexible licensing or related agreements, to further reduce the barriers to collaboration, skill-building and discovery.

Often UGC is partially or totally monitored by website administrators to avoid offensive content or language, copyright infringement issues, or simply to determine if the content posted is relevant to the site's general theme. However there has often been little or no charge for uploading user-generated content.

User-generated content marked a shift from media organizations creating online content to providing facilities for amateurs to publish their own content. It has also been characterized as 'Conversational Media', a two-way process of publishing content. Conversational or two-way media is a key characteristic of the so-called Web 2.0 which encourages the publishing of one's own content and commenting on other people's. The role of the passive audience therefore has shifted since the birth of the New Media, and an ever-growing number of participatory users are taking advantage of the interactive opportunities, especially on the Internet, to create independent content [4].

According to the Organization for Economic Co-operation and Development (OECD), the three central ideas for UGC are specified as follows:

- **Publication requirement:** While UGC could be made by a user and never published online, the focus is on the work that is published in some context, be it on a publicly accessible website or on a page on a social networking site, only accessible to a select group of people (e.g. fellow university students). This is a useful way to exclude email, a two-way instant message which is referred to an individual only.

- **Creative effort:** A certain amount of creative effort was put into creating the work or adding value to existing works. UGC also has a collaborative element to it, as in the case of website users editing collaboratively. If a user uploads his/her photographs, however, expresses his/her thoughts in a blog, or creates a new music video, this could be considered UGC. However the minimum amount of creative effort is hard to define and depends on the context.
- **Creation outside of professional routines and practices:** User generated content is generally created outside of professional routines and practices. It does not necessarily require the institutional or a commercial market context. In extreme cases, UGC may be produced by non-professionals without the expectation of profit or remuneration. Motivating factors include: connecting with peers, achieving a certain level of fame, notoriety, or prestige, and the desire to express oneself [4].

2.5 Citizen Journalism

Citizen journalism is also known as public, democratic or street journalism. This concept is for members of the public “playing an active role in the process of collecting, reporting, analyzing and disseminating news and information.”. The aim of this participation is to provide independent, reliable, accurate, wide-ranging and relevant information that a democracy requires. This concept is different from the one that is practiced by the professional journalism (community journalism or civic journalism), or the form of journalism whereby professional and non-professional journalists work together (collaborative journalism) [5].

The idea behind citizen journalism is that people without professional journalism training can use the tools of modern technology and the global distribution of the Internet to create, augment or fact-check media on their own or in collaboration with others. This spans from writing a city council meeting on a blog or online forum to fact-checking a newspaper article from the mainstream media and pointing out factual

errors or bias from it, or making a video record of a similar event and post it on a site, for example YouTube [6].

3 Application Development Environments

3.1 Overview

Android applications, like most mobile phone applications, are developed in a host-target development environment. In other words, one can develop an application on a host computer (where resources are abundant) and download it to a target mobile phone for testing and ultimate use. Applications can be tested and debugged either on a real Android device or on an emulator. For most developers, using the emulator is easier for initial development and debugging, followed by final testing on real devices [7].

Testing an application on the emulator is almost the same as testing it on a real device, except for some certain features that are limited on the emulator, such as video, which cannot be viewed properly like the real device, location-based application, which will require a GPX or KML file to simulate the movement of the device, and some few others. In developing an Android application, the required tools will be used to set up an appropriate development environment on a computer. Linux, Windows and OS X support the development environment. This project used Windows Vista as the development platform. The Android SDK supports several integrated development environments (IDEs), but for the purpose of this project, Eclipse was considered, due to the fact that it is best integrated with the Android SDK and it is free. Eclipse IDE, Sun's Java Development Kit (JDK), Android Software Developer's Kit (SDK) and Android Development Tool (ADT) are what is needed to set up the development environment.

3.2 Installations and Plug-in

For an efficient setup of the development environment, Android SDK requires JDK version 5 or version 6. This can be downloaded from the list of Java products.

Assuming that Eclipse is already installed properly in a computer, the Android SDK is now set to be installed. Android SDK is distributed through the Google web site. It needs to be reviewed and terms of the license need to be accepted to continue. Details of the installation are all specified depending on the platform to be used.

Platform	Package	Size	MD5 Checksum
Windows	android-sdk_r05-windows.zip	23449838 bytes	cc2c51a24a2f876e0fa652e182ef5840
Mac OS X (intel)	android-sdk_r05-mac_86.zip	19871714 bytes	6fcfeed0e1c36624c926551637eb3308
Linux (i386)	android-sdk_r05-linux_86.tar.gz	16208523 bytes	1d695d6a31310406f5d49092a1bd9850

Figure 1. Android sdk download [8].

Figure 1 shows various Android SDKs as specified on the Google website. Android SDK comes with some sample applications that would help developers acquaint themselves with the basic concepts of the programming. Developers can also build upon the sample application as an existing program. Sometimes, the best way to learn how things are done is to look at some sample codes. On the Android developers site the source code of some sample Android applications that are included in the Android SDK could also be downloaded. Each version of the Android platform available for the SDK includes a full set of sample applications (which may vary between different versions of the platform). [9]

3.3 Android Architecture

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

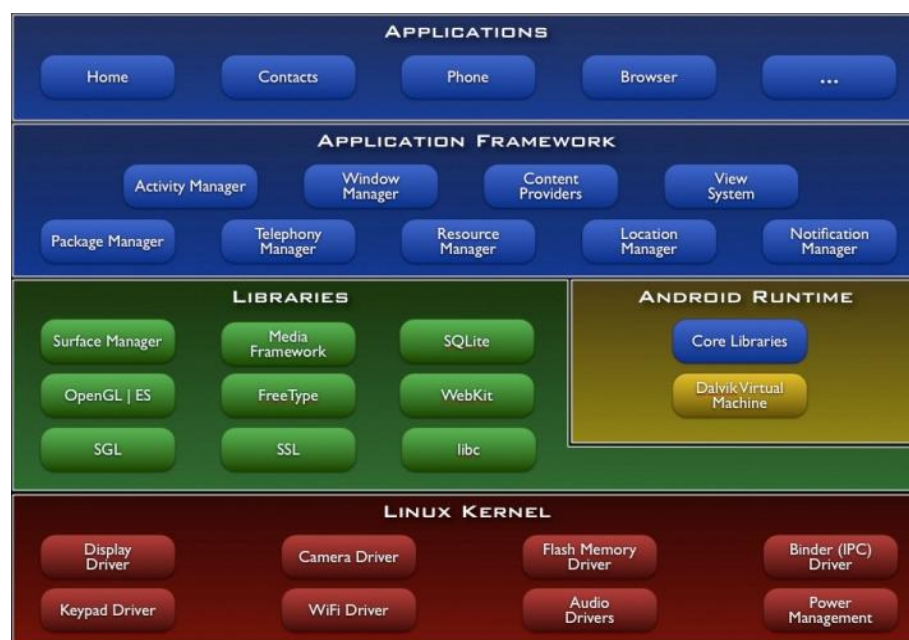


Figure 2. Android Architecture [10].

Figure 2 shows Android architecture from the Kernel level to the application level. This also includes a set of core libraries that provides most of the functionalities 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 designed so that multiple instances can efficiently run on a single device. It 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 kernel also acts as an abstraction layer between the hardware and the rest of the software stack [10].

Dalvik Virtual Machine

Dalvik is the name of Android's virtual machine. It was named by Bornstein after the fishing village of Dalvik in Eyjafjörður (Iceland), where some of his ancestors lived. As explained in section 3.3, it is an interpreter-only virtual machine that executes files in the Dalvik Executable (.dex) format, a format that is optimized for efficient storage and memory-map able execution. The virtual machine is register-based, and it can run classes compiled by a Java language compiler that have been transformed into its native format using the included "dx" tool. The VM runs on top of a Linux 2.6 kernel, which it relies on for underlying functionality (such as threading and low level memory management). The DalvikVM was also optimized to be running in multiple instances with a very low memory-footprint. Several VMs protect ones application from being dragged down by another crashed Application [11]. Unlike Java virtual machine (Stack-based Virtual Machine) found in other desktop computers, DalvikVM is register-based virtual machine which suits well for mobile processors. Besides, registered-based VM provides faster execution time.

3.4 Android Application Life Cycle

An application life cycle consists of the steps that the processes of the application must follow from execution to termination. Every application, regardless of the language it was written in, has a specific life cycle, and Android applications are no exception. All of the running processes are watched by Android and, depending on how the activity is running (that is, a foreground activity, background activity, and so forth), Android may choose to end the activity to reclaim needed resources. The cycle starts when an Android application is created, the processes are started, events are fired, processes are stopped, and the application is destroyed [12].

Unlike most traditional environments, Android applications have no control over their own life cycles. Instead, application components must listen to changes in the application state and react accordingly, taking particular care of being prepared for untimely termination. The Android application runs in its own process, this implies that it runs a separate instance of Dalvik. Memory and process management of each application is handled exclusively by the runtime. Android aggressively manages its resources, ensuring that the device remains responsive. This means that processes (and their host applications) will be killed, without warning if necessary, to free resources for higher-priority applications, generally those that are interacting directly with the user at the time [13, 50].

In Android platform, applications are run as a separate Linux process. So the application lifecycle is closely related to the process lifecycle. The application process lifecycle is handled by the system depending on the current system memory state. In the case of low memory, the system kills some less important process. The process importance is decided depending on the state of the process components [14]. The process at which the user is currently using is the “Foreground process”. This process is at the top of the stack in the activity. It begins in an activity when the “onResume ()” method is been called. This priority could be giving to a Broadcast Receiver by passing execution to the “onRecieve ()” method and also to a service by executing callback functions such as “onCreate ()”,” onStart () “or “onDestroy()”.

A visible process is one that does not have any foreground components, but still can affect what the user sees on the screen. A process is considered to be visible if it hosts an activity that is not in the foreground, but is still visible to the user (its `onPause()` method has been called). This may occur, for example, if the foreground activity is a dialog that allows the previous activity to be seen behind it, and also if it hosts a service that is bound to a visible activity. A visible process is considered extremely important and will not be killed unless doing so is required to keep all foreground processes running. [15].

Provided there is enough memory to accommodate other processes such as the foreground and visible processes, the service process could be used to do other things that are useful to the user, such as downloading files from the Internet or playing music while the user is busy interacting with other process. This works in the system in a way that the user does not see it. An empty process is one that does not hold any active application components. The only reason to keep such a process in the cache is to improve startup time the next time a component needs to run in it. The system often kills these processes in order to balance the overall system resources between process caches and the underlying kernel caches [15].

A background process holds the activity that is not currently visible to the user. This implies that the `onStop()` method of such an activity has been called. In order to reclaim memory for other processes, the background process can be killed. Since there could possibly be many background processes running at a time, an LRU (least recently used) list is made to ensure that the most recently viewed activity process is to be the last to be killed.

3.5 Android Application Framework

Android provides an open development platform that gives developers the opportunity to take advantage of for example the device hardware, access location information, run background services, add notification to the status bar, among others. Since full access to the framework APIs is given to developers and the application architecture is designed to simplify the reusability of the components, any application can publish its capabilities. Any other application may then make use of those capabilities (subject to security constraints enforced by the framework) [10].

3.5.1 Activity Manager

Activity Manager is a project management tool that is simple to use, lightweight, very efficient and customizable. It features collaborating repository administration, tasks repository administration, contributions management (activity management), and an extensible report facility (with built-in templates). It allows one to build and maintain a hierarchical task tree [16]. In Android the activity manager manages the life cycle of the application. This is managed as an activity stack when a new activity is started, it is kept on top of the stack and it becomes the running activity.

The essential states are specified as follows:

- If an activity is on the foreground of the screen (at the top of the stack), it is active or running.
- If an activity has lost focus but is still visible (that is, a new non-full-sized or transparent activity has focus on top of your activity), it is paused. 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 if the system is in extreme low memory situations.

- If an activity is completely obscured by another activity, it is stopped. 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 the activity from memory by either 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 [17].

Figure 3 below shows the Activity life cycle.

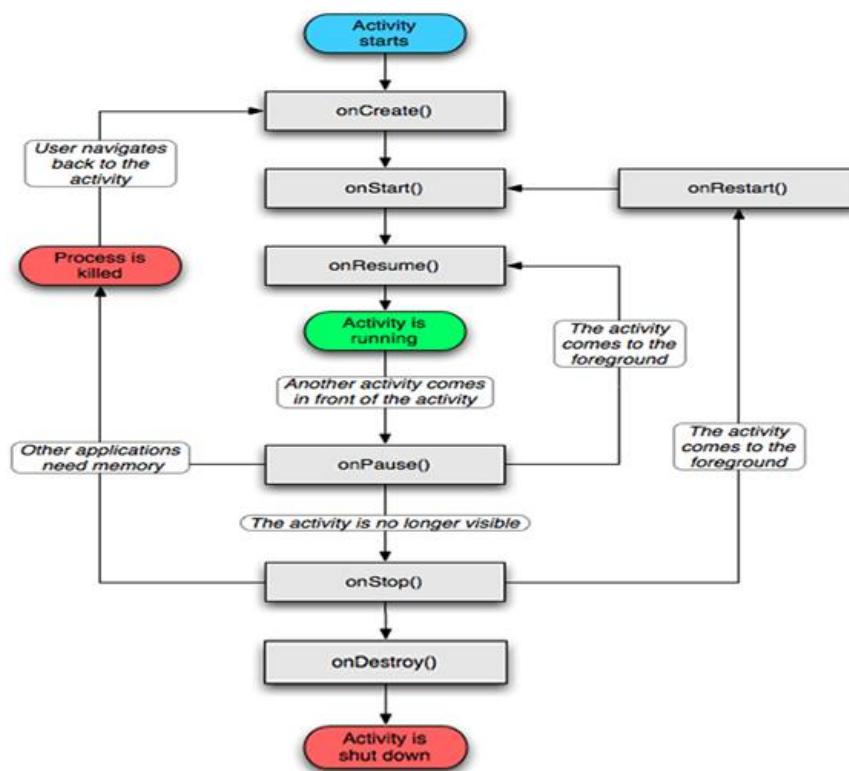


Figure 3. Activity life cycle [17].

Figure 3 shows the flow chart of an activity life cycle from the point of start to the point of termination.

3.5.2 Content Provider

In the Android application, data can be stored in files, in the SQLite database or in an other mechanism. Classes that implement a standard set of methods letting other applications store and retrieve the type of data they handle are known as content providers. This is useful when there is need for an application to share data with other applications [11, 18].

In view of providing an avenue for data to be shared across an application, Android ship a number of content provides for common data types such as audio, video, images, and personal contact information. They can be queried for the content they hold or they can be modified, and it is also possible to create once own content provider. In any case some content providers demand that proper permission be taken before they can be accessed [18].

3.5.3 Resource Manager

The resources required in an application development are handled by the Resource Manager. The Application resources are stored under the “res” folder of the project hierarchy. In this folder, each of the available resource types can have a subfolder containing its resources. There are seven primary resource types that have different folders:

- Simple values
- Drawables
- Layouts
- Animations
- XML
- Styles
- Raw resources

When the application is built, these resources will be compiled as efficiently as possible and included in the application package. This process also creates an R-class file that contains references to each of the resources that are included in the project. This helps for an easy reference to the resources in the code, with the advantage of design time syntax checking [13, 53].

3.5.4 Location Manager

The possibility for Android to access the Google Maps infrastructure is one of the strengths of the platform. Location-based services in Android works exactly as one would expect them to work with only minor exceptions. Android allows developers to specify which location lookup method to use. This allows one to customize the power consumption, cost, and accuracy based on the specific use one has in mind for the application [19, 75].

Android's location-based services are based on two pillars: the map APIs which are contained in the "*com.google.android.maps*" package, and the location APIs which are in the "*android.location*" package. The map API provides the facilities to display and manipulate a map such as zooming, changing the mode of the map for instance, street view, satellite view and traffic view. This project uses the location API to extract the longitude and latitude coordinates of the device's position and incorporate it into the Geocoder class to obtain the address of a specific location.

Geocoding is a concept whereby latitude/longitude pair is converted into an address or location, and the "*android.location.Geocoder*" class provides this facility. The class provides both forward conversion (converting a given address to its latitude and longitude pair) and backward conversion (translating a latitude and longitude pair into a list of addresses) [20,249].

3.5.5 Notification Manager

Nowadays cell phones do not only serve the purpose of making phone calls alone but also as a personal assistant. It has now reached the level where every feature in a computer is inclusive in a mobile phone. With all these applications running on mobile phones, there is need for an application to notify the users of some certain events or to get the use's attention [21,192]. In Android programming, the notification comes in different ways, such as a pop up message indicating the arrival of a new e-mail or SMS, weather notification, flashing LED, or vibration of the phone. There is also persistence notification that shows in the status bar until the user attends to it, among others. All these actions are represented by the notification class [21,196].

4 Media API and Networks

4.1 Overview

The ability to produce a video file and also to communicate with the server using the Android technology are the focus of this chapter. Figure 2 on page 17 showed the media framework, which is one of the libraries that Android uses to support its exciting functionalities. Until the time of writing this, Android supports the following multimedia formats: JPEG, OGG, Mpeg4, PNG, Bitmap, MP3 and 3GPP. Most Android applications such as the media application requires that the developer obtain permission to use some facilities, such as permission for audio recording (`<uses-permission android: name="android.permission.RECORD_AUDIO"/>`), or permission for video recording (`<uses-permission.android: name =" android.permission. RECORD_VIDEO"/>`). This is done in the manifest file of the application.

Figure 2 also showed the Wi-Fi driver, which is situated in the Linux kernel. This creates the possibilities for this project work to communicate with the server through the wireless connection and send files and other metadata across. More details of the practical implementation will be given in chapter 5.

4.2 Using the Media API

Android offers a fairly straightforward way in accessing the media platform. It has built-in encoding/decoding of some common media types. Audio and video can be played from different types of data source, such as playing audio or video files from media files stored in the resources folder, stand-alone files in the file system or from a data stream arriving over a network connection. Data streaming, refers only to files that support streaming to Android device. The MediaPlayer Class is the class used for implementing this process. The platform also provides the possibility to record audio and video too with the help of the MediaRecorder class [22].

4.3 Media Player

The MediaPlayer is easy to use. For example if a supported audio file is kept in the res/raw folder in a project, that file can be found by using the R-class to make reference to it. Then an instance of the MediaPlayer is made and the file is parsed to the “MediaPlayer.create” method, the create method loads the audio file from the resource, next the prepare method is called, then the start method. Figure 4 shows the code snippet.

```
MediaPlayer myMediaPlayer= MediaPlayer.create(context, R.raw.the_supported_file);
myMediaPlayer.prepare();
myMediaPlayer.start();
```

Figure 4. Playing from a raw file

Playing a media file from the file system directly is almost similar to playing it from the resource folder. The MediaPlayer instance would call the setDataSource method, which takes a string that point to the file to be played. It also has an overloaded version that can be used to customize the data source to a specific need [20,306]. Figure 5 below shows a sample format.

```
MediaPlayer myMediaPlayer = new MediaPlayer();
myMediaPlayer.setDataSource(Path_to_the_supported_file);
myMediaPlayer.prepare();
myMediaPlayer.start();
```

Figure 5. Using the setDataSource method

Playing video files is more involved with the media player than playing audios. Since the surface view for the video has to be provided. Android provides a VideoView widget that handles that. This view can be used in any layout manager and it provides a number of display options [21, 245]. Chapter 5 gives a more detailed explanation of how this was used in this project.

4.4 Media Recorder

Multimedia recording is handled by the “MediaRecorder class “. In the Android application recording audio or video can be done by creating an instance of the MediaRecorder, configuring the video and audio sources (generally the camera and the microphone), output format, video size, frame rate, and the video and audio encoder to be used [13,318].

```
MediaRecorder myMediaRecorder = new MediaRecorder();  
myMediaRecorder.setAudioSource(MediaRecorder.AudioSource.MIC);  
myMediaRecorder.setOutputFormat(MediaRecorder.OutputFormat.DEFAULT);  
myMediaRecorder.setAudioEncoder(MediaRecorder.AudioEncoder.DEFAULT);  
myMediaRecorder.setOutputFile("path_to_the_sdcard"+"filename.mp3");  
myMediaRecorder.prepare();  
myMediaRecorder.start();
```

Figure 6. Sample audio recorder

Figure 6 above configures the Media Recorder so that it records audio from the microphone and encodes it using the default format. The “setOutputFile” method sets the new file called filename.mp3 to the “SDcard”. Chapter 5 showed more detailed explanation.

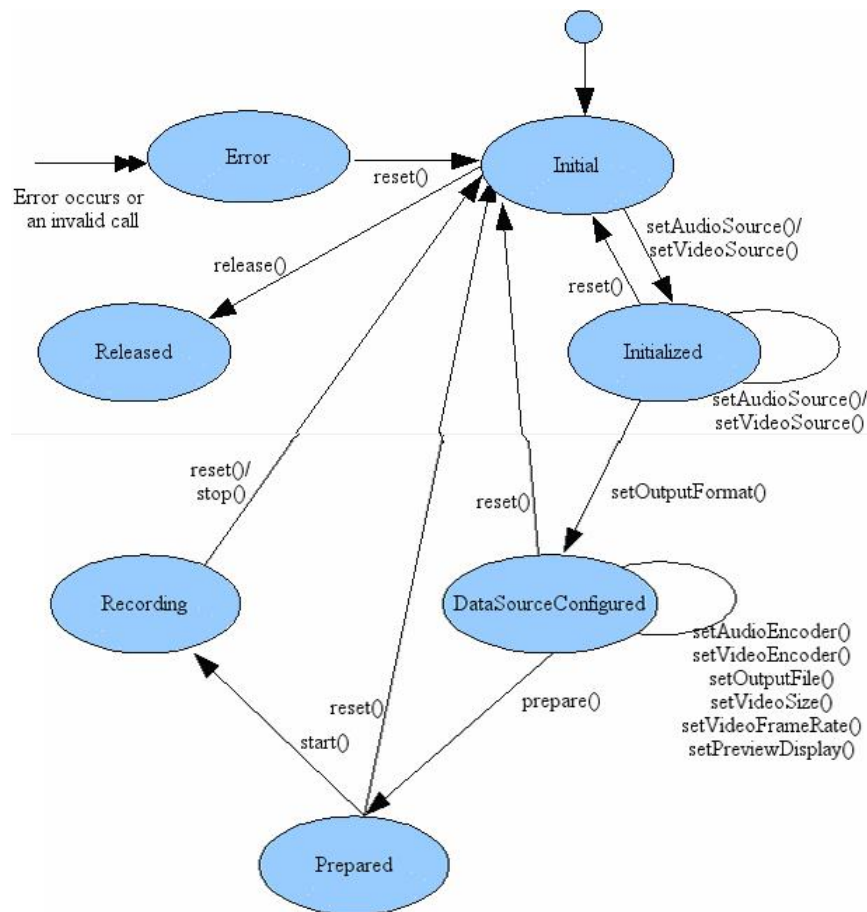


Figure 7. Media Recorder state diagrams [23].

Figure 7 shows the media recorder state diagram which gives an idea of how the audio/video recording methods relate to one another. Video recording was not possible with the older version of the SDK until the advent of version 1.5. Capturing video using the media framework began with the 1.5 version. This version demands that the video recording application be tested on a real device because the required hardware-encoder module is not available in the version's emulator [20,385].

4.5 WLAN Connectivity

The Android platform, like most modern mobile device platforms, offers developers a wide range of ways to make use of the Internet access. Some offer high-level access, such as the integrated WebKit browser component. It is also possible to use a raw socket or to leverage APIs (both on-device and from 3rd party JARs) that give access to specific protocols such as HTTP, XMPP, and SMTP. The three main connection techniques for Internet connectivity provided by Android are offered transparently to the application layer:

- GPRS, EDGE, and 3G: Mobile Internet access provided by carriers that offer mobile data plans.
- Wi-Fi: Wi-Fi receiver and mobile hotspots are becoming increasingly common.

It is compulsory to obtain “Permission for the internet” from the application manifest file for any application that would make use of the Internet [13,142; 24,207]. The basic pattern for opening an Internet data stream is shown in figure 8 below.

```

try {
    URL url = new URL("the_url_to_connect_to");
    URLConnection myCon = url.openConnection();
    HttpURLConnection myhttpCon = (HttpURLConnection) myCon;
    int responseCode = myhttpCon.getResponseCode();
    if (responseCode == HttpURLConnection.HTTP_OK) {
        InputStream in = myhttpCon.getInputStream();
        // do some thing with the input stream here
    }
}
catch (MalformedURLException e){
    // do malformed url exception catch here
}
catch (IOException e){
    // do input/output catch here
}

```

Figure 8: Basic network connection

Better form of java-related connection to a file in the server could be used to replace this if there is need.

4.5.1 Broadcast Receiver

In Android, the message-passing mechanism that lets one declare an intention for an action to be performed is known as Intent. Intent can be used to support interaction between any of the application components available on an Android device, no matter which application they belong to. It can be used explicitly, by using Intent to start a new Activity, which is implemented by calling the class to be loaded or implicitly, by requesting an action to be performed on a piece of data.

Intent can also be used to broadcast messages across the system. For example Android uses broadcast Intent to announce system events, such as changes in the Internet connection status or battery charge levels [13,114]. In monitoring Wi-Fi connectivity, the Wi-Fi manager broadcasts Intent whenever the connectivity status of the Wi-Fi network changes [13,348]. To receive the intents and using them is done in the Broadcast Receiver [25]. This implies that in a given developer's intention to implement an action, there has to be a registered receiver that listens to and puts such an action into use.

4.5.2 Background Services

A Service is code that is long-lived and runs without a UI. A good example is a media player playing songs from a play list. In a media player application, there would probably be one or more activities that allow the user to choose songs and start playing them. However, the music playback itself should not be handled by an activity because the user will expect the music to keep playing even after navigating into a new screen. In this case, the media player activity could start a service using `Context.startService()` to run in the background to keep the music going. The system will then keep the music playback service running until it has finished. However one can connect to a service (and start it if it is not already running) with the `Context.bindService()` method. When connected to a service, one can communicate with it through an interface exposed by the service [11, 18].

This project made use of a process that runs in the background, by connecting and writing a specified file and strings to the server while protecting the foreground with another thread that runs the progress bar. Chapter 5 explains the practical implementation.

4.5.3 Communication Protocols

Every mobile provider supports both voice and data networks, of one or more types. A data network is one of the interesting parts for Android-enabled devices: the ability to link the available data on the Internet to applications and vice versa. It is the combination of a platform, hardware capability, software architecture and access to network data that makes Android more interesting.

Android provides several ways to access network data: mobile IP networks, Wi-Fi, and Bluetooth. A network is a group of interconnected computers. Networking has grown from something that was once available to governments and large organizations only to an amazing Internet linked to hand-held devices that we can carry about in our pockets and handbags. The basic idea behind a network is that data is sent between connected devices with particular addresses. Connections can be made, for example, over wire, and radio waves. Each addressed device is known as a node and each node can be a mainframe, a PC, or any other device with a network stack and connectivity, such as Android-enabled devices. Nodes in a TCP/IP network are identified by their IP addresses. Each protocol in this family has a specific role; these roles are often described as layers.

A server socket is a stream that developers can read or write raw bytes to, at a specified IP address and port. This allows one deal with data, without worrying about media types, packet sizes, and so on. This is yet another network abstraction intended to make the job of the programmer easier. The philosophy that everything should look like file I/O to the developer, comes from the early UNIX days but has been adopted by all major operating systems in use today such as Windows, Mac, and Linux [21,150-155].

In order to take advantage of the existing server products to send data, this project used a web server and leverage HTTP. It is a stateless protocol that involves several different methods that allow users to make requests and get responses from the server. PHP, an efficient and flexible server side script was also used here for a reliable productivity.

4.5.4 Requirements to Port Files into YouTube

YouTube is a video sharing website in which users can upload, share, and view videos. Three former PayPal employees created YouTube in February 2005. In November 2006, YouTube, LLC was bought by Google Inc. for \$1.65 billion, and is now operated as a subsidiary of Google. The company is based in San Bruno, California, and uses Adobe Flash Video technology to display a wide variety of user-generated video content, including movie clips, TV clips, and music videos, as well as amateur content, such as video blogging and short original videos. Most of the content on YouTube has been uploaded by individuals, although media corporations including CBS, BBC, UMG and other organizations offer some of their material via the site, as part of the YouTube partnership program [26].

This project work was to create videos with Android mobile phone and upload them to a dedicated server for further uses. One of the uses of the uploaded file could possibly be to re-direct them to other video sharing websites. YouTube offers several possibilities to upload files to their server. In a situation where files are already seating in a server, the best way of sending them to YouTube is by the use of “Resumable uploads”. It has the ability to continue the file transfer from where it stopped, in the case of any interrupt while uploading to YouTube.

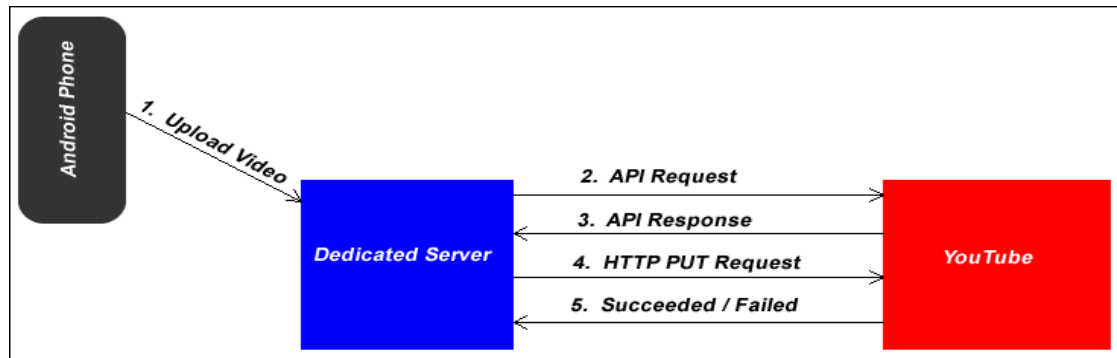


Figure 9. Data flow to YouTube

When a file is uploaded from the phone to the dedicated server, the server side script sends an API request to upload video to YouTube. This request contains the video filename and the metadata of the video.

```

POST /resumable/feeds/api/users/default/uploads HTTP/1.1
Host: uploads.gdata.youtube.com
Authorization: AuthSub token="<authentication_token>"
GData-Version: 2
X-GData-Key: key=<developer_key>
Content-Length: <content_length>
Content-Type: application/atom+xml; charset=UTF-8
Slug: <video_filename>

API_XML_request
  
```

Figure 10: Example format of an API request.

Figure 10 is an example of the API request that uses the AuthSun authentication scheme for resumable uploading.

- `authentication_token`:- This value contains the authentication token for the request. The token will either be a ClientLogin token, an AuthSub single-use token, an AuthSub session token or an OAuth access token.
- `developer_key`:- This value uniquely identifies the application that is submitting the request to upload the video. This is received by registering a Google Account.

- `content_length` :- This value contains the length (in bytes) of the entire body of the HTTP POST request.
- `Video_filename` :- This value contains the file name of the source video that one is uploading.
- `API_XML_Request` :- This value contains information about the uploaded video file in the form of an Atom XML entry, such as the media: group, title category, keywords.

```
HTTP/1.1 200 OK
Location: http://uploads.gdata.youtube.com/resumableupload/AF8GKF...0Glpk0Aw
Date: Fri, 04 Dec 2009 16:14:30 GMT
Pragma: no-cache
Expires: Fri, 01 Jan 1990 00:00:00 GMT
Cache-Control: no-cache, no-store, must-revalidate
Content-Length: 0
Content-Type: text/html
```

Figure 11. Sample API response to a request

Having sent the API request to YouTube server, the API returns an HTTP response that contains a location header, which identifies the URL that will be used to upload the actual video file. This is shown in figure 11. The URL must be extracted and used for the actual sending process.

```
PUT <upload_url> HTTP/1.1
Host: uploads.gdata.youtube.com
Content-Type: <video_content_type>
Content-Length: <content_type>

<Binary_file_data>
```

Figure 12. Format for video uploading to Youtube

Figure 12 shows the format on how to upload the actual file. The highlighted variable must be provided to the HTTP PUT request. The following list explains how to populate each value:

- `upload_url` :- This value is the URL that was extracted in step 2 from the `Location` header of the API response for the metadata upload request.
- `video_content_type`:- This value specifies the MIME type of the uploaded video file. The MIME type can be a video media type, such as `video/mpeg` or `video/mp4`, or it can be `application/octet-stream`.
- `content_length`:- This value specifies the length, in bytes, of the entire body of the HTTP PUT request.
- Binary File Data: - This value contains the binary code for the video file that is being uploading.

Completing the upload process

When a request to upload the video content for a resumable upload is submitted, one of two scenarios could occur:

- If the upload completes, then the API returns a response indicating whether the upload succeeded or failed.
 - For a successful upload, the API returns an Atom entry that contains information about the uploaded video.
 - For an unsuccessful upload, the response contains an error response that helps to explain the cause of the upload failure.
- If the upload fails because the network connection between the dedicated server and the YouTube server, then the server-side application will not receive an API response. In this case, one may be able to resume the upload from the point of the interruption [27].

5 Reporter's Video Recorder Based on Android

5.1 Overview

The goal of this project was to develop an easy-to-use Android-based video recorder that would be capable of collecting the required meta-data automatically and upload the data to a server with the availability of WLAN. It is a prototype that can be customized to the need of organizations that require videos of events that took place in different geographical locations.

The application also triggers the system to show the user the available wireless network connections at any given location that mobile phone is used. The meta-data, such as the file name, file category, date, present location address, user name and others are sent to the server along with each file. Geo-coding was used to convert the latitude and longitude coordinates into a specific address.

5.2 Requirements

What is required of the application was to ensure that a video file is created and sent along with some necessary data to the server efficiently, and also the possibilities of downloading video files from the server. These requirements are stated as follows:

- An easy to use Android-based video recorder
- Collection of the required meta-data automatically (filename, category, date and time, and present location address).
- Using Geo-coding to convert latitude and longitude coordinates into a specific address.
- Uploading the video file and metadata to a server with the availability of WLAN
- Database containing the details of all the files sent
- Downloading the video file from the server.

5.3 Design


Based on the requirements specified, the application used the Android's "MediaRecorder" class and its vital methods, surface view and a toggle button. Combining these components together logically, and obtaining a permission to use the system's camera helped to achieve the goal. The video file that is to be created would be stored locally with a file name in the SDcard. Since other documents were possibly kept in the SDcard of a mobile phone, the Java "FileNameFilter" interface was implemented to filter all the files with the extension name ".mp4". The "System.currentTimeMillis()" method will be used to extract the current date and time which a file is sent. Android has software framework called the location manager, which is capable of extracting the mobile phone's longitude and latitude coordinates with its "getLongitude()" and "getLatitude()" methods respectively. These coordinates will be parsed into the Android's Geocoder class which built strings of address from them.

All the metadata will be extracted as strings and parsed to the "URLConnection" class, to be sent to the server. Before the application can connect to the network, the mobile phone WLAN will be activated. In such a case the application will parse "ACTION_WIFI_SETTINGS" to an intent variable, to open the system's network settings. This will let the user choose from the list of available hotspots and enter the user name and password. A PHP script will be used to handle the file and metadata in the server side. This will be implemented so that feedback will be sent back to the phone to notify the user whether the sent file was successful or not. If successful, the file will be kept in a directory and the metadata will be saved in the database.

Another PHP script will be written to roll out all the file names in the database and their sizes. The file will also use a hyperlink to point each file name to the directory where they are saved. The URL path of the PHP file will be used by the downloading part of the mobile application. Using Android's "WebViewClient" to show the file on the phone, the "InputStream" class to read byte data from the connection and the "FileOutputStream" subclass to write the data to the SDcard will help to achieve the set goals.

Table 1 shows details of the functional units of the application. It specifies part the of the program where users can communicate with the internal functionalities of the application to achieve specific results.

Table 1. Descriptions of the system functionalities

(1) Application Icon	 <p>This icon was designed using Adobe Fireworks and saved as .png file in the drawable folder of the application. The icon has to be clicked to start the application.</p>
(2) Application Menu	<p>Record Video: This menu leads to the video recording part of the application.</p> <p>Play Video: Click on this button to play recorded videos from the sdcard.</p> <p>Send Video: To send file to the server.</p> <p>Record Audio: This button takes you to the audio recording part of the application.</p> <p>Play Audio: To listen to the recorded audio from the sdcard.</p> <p>Delete Video: - For deleting videos.</p> <p>Search for WiFi: Used to search for wireless connection.</p> <p>Download File(s): Used for downloading file from the server to the sdcard.</p>
(3) Video recording interface	<p>Start/Stop buttons. Save button. Delete button.</p>
(4) Video/Audio play interface	<p>Video view (for video only). Play button. Stop button. Pause button. Back button. Forward button.</p>

(5) Sent to server interface	Select file from file list. Click on file.
(6) Delete file interface	Select file from file list. Click on file. Accept to delete or cancel deleting.
(7) Download File(s)	Select file from the web view. Click on file. Click on “Download” button.

Table 1 gives the basic understanding of how the application can be used. The icon appears where all the programs in the device are kept. Clicking on it triggers the application to be launched. This brings the user to the main Activity that contains the menu that leads to other Activities. From this point users can navigate to any of the views specified in table 1.

In Android, all applications are said to be of the same hierarchy. Figure 13 below shows the internal system view of this application. The Activities have their different purposes. In this application videos are recorded straight to the device “sdcard”. Files can be deleted from the “sdcard” if needed. They can also be downloaded from the server to the “sdcard”. Videos can also be played or sent to the server. The files have to pass through the file filter mechanism to the list of files to be sent to the server or to the play list.

The file filter mechanism is important because the user also has other file formats stored in the “sdcard”. However this application only focuses on dealing with files that has the extension “ .mp4 ”. To ensure that the only file the application gets from the “sdcard” is .mp4 file format, the filtering mechanism has to loop the “sdcard” for selection.

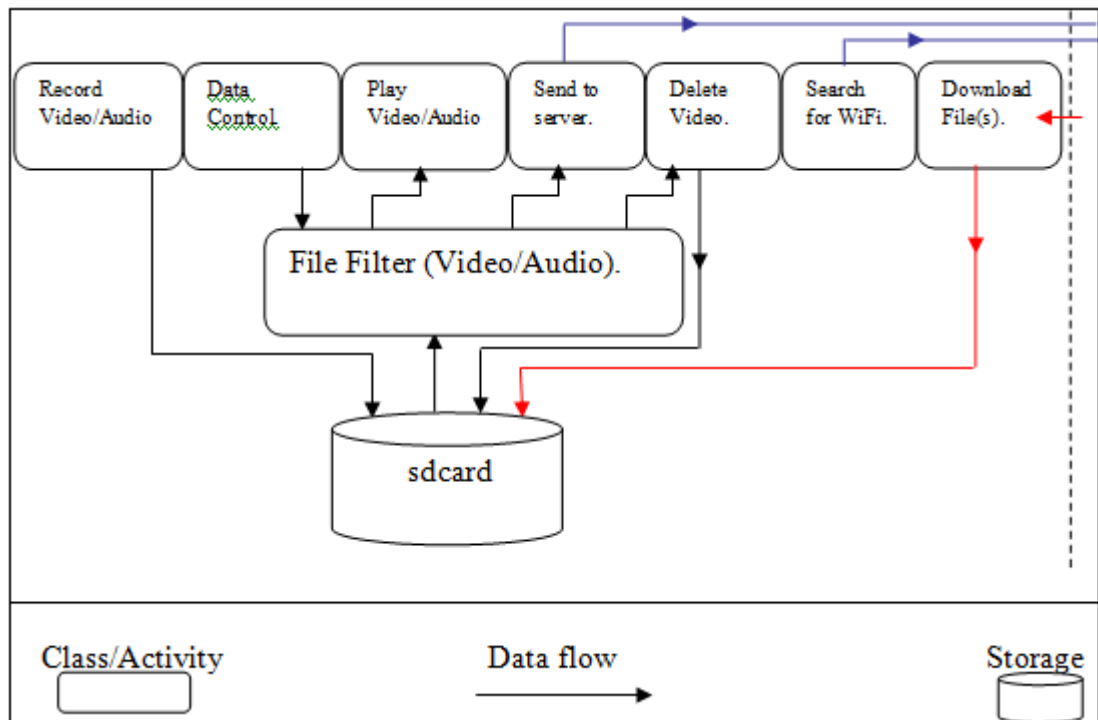


Figure 13. System Architecture

Figure 13 shows the flow of data from the sdcard to the various activity classes, and the flow of data from the server-side to the sdcard.

5.4 Implementation

5.4.1 Application Menu

To create a menu in an application, “Import android.view.Menu;” has to be added to the Activity. The onCreateOptionsMenu() is a Boolean method that takes the Menu variable as a parameter. It is called when a button is pressed. As the name implies, menus in Android are also known as options menus. To populate the menu with system menu items, the base-class of this method has to be called, thereby using the parameter to call the add() method. The add() method is of the form “param.add(<Group>,<item id>,<order>,<”title”>);” . Figure 14 shows a sample of the application menu.

```

public boolean onCreateOptionsMenu(Menu menu) {
    super.onCreateOptionsMenu(menu);
    menu.add(0, 0,0, "Record Video");
    menu.add(0, 1,1, "Play Video");
    menu.add(0, 2,2, "Send Video");
    menu.add(0, 3,3, "Record Audio");
    menu.add(0, 4,4, "Play Audio");
    menu.add(0, 5,5, "Delete Video");
    menu.add(0, 6,6, "Search for WiFi");
    menu.add(0, 7,7, "Download File(s)");
    return true;
}

@Override
public boolean onOptionsItemSelected(MenuItem item){

    switch (item.getItemId()) {
    case 0:
        toRecordVideo();
        return true;
    case 1:
        toReportersVideoPlayer();
        return true;
    case 2:
        toReportersUploadVideo2();
        return true;
    case 3:
        toRecordAudio();
        return true;
    case 4:
        toReportersAudioPlayer();
        return true;
    case 5:
        toDeleteVideo();
        return true;
    case 6:
        toWlan();
        return true;
    case 7:
        toDownload();
        return true;
    }
    return true; //its important to return true to see menu.
}

```

Figure 14. Application menu implementation

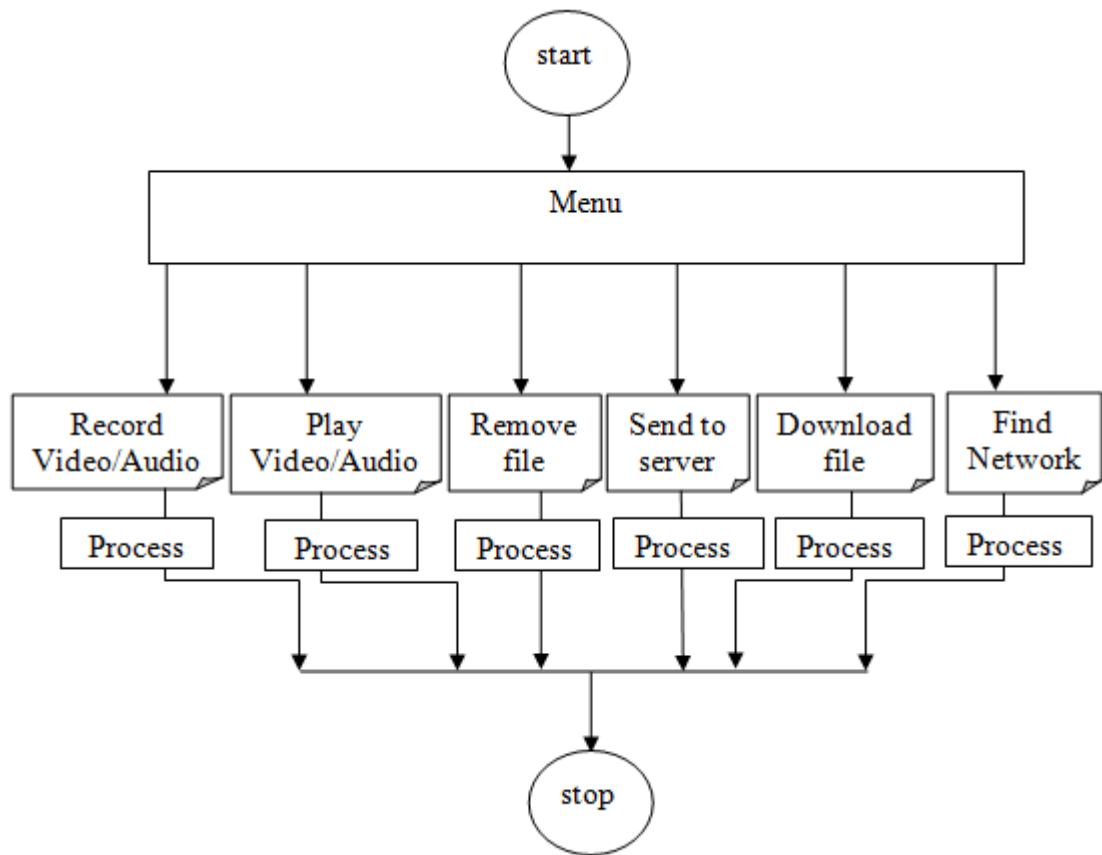


Figure 15. External system view

Figure 15 illustrates how the various Activities function separately within the application. Android can possibly run more than one process at a time, but due to the limited screen size compared to a desktop computer, one Activity is shown at a time. An Activity can be seen as a webpage. Here it is possible to play the audio file at the background while using another Activity. An Activity can link to another Activity using “Intent”. This depends on what the application is designed to do. In this application the “Record Video” Activity is also implemented so that when the stop button is pressed, it moves to another Activity where the user decides whether to save or delete the file. Sending a file to the server, playing video files, and downloading files has various ways of linking between Activities for efficient control of data.

5.4.2 Video Recorder

A video recorder in Android requires principles to follow to achieve success. It is implemented by creating a “SurfaceView” in an xml file (recordeaudio.xml) and linked to the Activity using the “findViewById()” method. The SurfaceView provides a dedicated drawing surface embedded inside the view. It allows customization such as controlling the format and resizing. It provides a hole in the window to allow its surface to be displayed. A window is required to fit the SurfaceView, which is contained in the “android.view.Window” package. Some features in the window can be altered, such as the “full screen”, the “no title bar” among others. Figure 16 below shows a code snippet that adds a window to the application. The window is set not to have a title, and the orientation is set to landscape, which has a full screen and allows translucency.

```
requestWindowFeature (Window.FEATURE_NO_TITLE) ;  
setRequestedOrientation (ActivityInfo.SCREEN_ORIENTATION_LANDSCAPE) ;  
getWindow().setFormat (PixelFormat.TRANSLUCENT) ;  
getWindow().setFlags (WindowManager.LayoutParams.FLAG_FULLSCREEN,  
WindowManager.LayoutParams.FLAG_FULLSCREEN) ;
```

Figure 16. Sample code for the video window

As mentioned in chapter 3, a video recorder can be implemented using the MediaRecorder class, which is found in the “android.media.MediaRecorder” package. It is used to configure some media file features such as the video and audio encoders, video and audio source, video size, output format and frame rate. Figure 17 below shows how the application uses this class.

```

MediaRecorder myMediaRecorder;
myMediaRecorder = new MediaRecorder();
myMediaRecorder.setAudioSource(MediaRecorder.AudioSource.MIC);
myMediaRecorder.setVideoSource(MediaRecorder.VideoSource.CAMERA);
myMediaRecorder.setOutputFormat(MediaRecorder.OutputFormat.MPEG_4);
myMediaRecorder.setAudioEncoder(MediaRecorder.AudioEncoder.DEFAULT);
myMediaRecorder.setVideoEncoder(MediaRecorder.VideoEncoder.MPEG_4_SP);
myMediaRecorder.setVideoSize(320,240);
myMediaRecorder.setVideoFrameRate(15);

```

Figure 17. Sample code for MediaRecorder class

The layout in the .xml file that contains the SurfaceView has to be set to the Activity, which was done by using “setContentView(R.layout.<the_xml_file>)”. In this case, linking the SurfaceView to the Activity, as said above, can be implemented by creating an instance of SurfaceView, which takes the “id” of the SurfaceView in the “.xml” file. The format used is given in figure 18 below. It links the xml file and the SurfaceView to the Activity and also keeps the surface on, through out the duration of the process.

```

setContentView(R.layout.<the_xml_file>);

SurfaceView mySurface;
mySurface=(SurfaceView) findViewById(R.id.<xml_file's_SurfaceView_id>);
mySurface.setKeepScreenOn(true);

```

Figure 18. Linking xml file to the Activity

Implementing Callback

The “Callback” is important for an Android video application. Having created an instance of SurfaceHolder, “SurfaceHolder.Callback” is used to get information about changes made in the surface. To let the Activity manage the “Callback” it has to be specified as follows:

- public class <activity name> extends Activity implements Callback{ };

The “Callback” interface implements three methods which play their roles as the application is set “on”:

- `surfaceChanged ()`:- This is called when there are changes made in the surface, such as size or the format of the surface.
- `surfaceCreated()` :- The surface is created in the first instance, thereby calling this method.
- `surfaceDestroyed()` :- This method is called whenever the surface is destroyed.

For simplicity purposes these methods also contains other methods that are involved in successfully handling the data flow. For example the “`prepare()`” and “`setOutFile(the_filepath)`” methods are kept in the `surfaceCreated()` method. This application must obtain permission for the camera to function.

5.4.3 Audio Recorder

An audio recorder, like a video recorder, uses the “`MediaRecorder`” class. It is simpler to implement compared to the video recorder because it does not require some complex aspects like the `SurfaceView`, `Callback` interface and its methods. This project implemented audio recorder using two buttons (Stop and Start buttons) to operate the application, it also shows the status of the operation while in use. Figure 19 below shows the application audio recording view.



Figure 19. Audio Recorder view

Figure 19 is used by pressing the “Start” button, which changes the Status from “Not Recording” to “Recording”. At this point the application gets sound from the user through the device microphone to the “sdcard”. Clicking the “Stop” button while the recording is on triggers the “stop()” method. This prompts the recording to be stopped and the status automatically changes.

```

MediaRecorder myMediaRecorder;

private void beginRecording() throws Exception {
kill_myMediaRecorder();
File outFile = new File(OUTPUT_FILE);
if (outFile.exists()) {
    outFile.delete();
}

myMediaRecorder = new MediaRecorder();
myMediaRecorder.setAudioSource(MediaRecorder.AudioSource.MIC);
myMediaRecorder.setOutputFormat(MediaRecorder.OutputFormat.DEFAULT);
myMediaRecorder.setAudioEncoder(MediaRecorder.AudioEncoder.DEFAULT);
myMediaRecorder.setOutputFile(OUTPUT_FILE);
myMediaRecorder.prepare();
myMediaRecorder.start();
setStatus("Recording ...");
}

private void stopRecording() throws Exception {
if (myMediaRecorder != null) {
    myMediaRecorder.stop();
    setStatus("Stopped Recording!");
}
}

```

Figure 20. Sample code for audio recorder

From the sample code in figure 20, “beginRecording()” method is used to start the audio recording proceedings. It starts by calling the “kill_myMediaRecorder()” method, which checks if the “myMediaRecorder” is equal to null or not. If it is, then the “Release()” method is called to free that instance. The next step is to check if the given file name already exist; if it does, the existing file will be deleted from the sdcard. Then the vital methods for the MediaRecorder class are called. The “setAudioSource()” method is used to set to MIC, the “setOutputFormat()” method is set to DEFAULT, the “setAudioEncoder()” method is set to DEFAULT, the “setOutputFile()” method is set to “.mp3” file extension, at this point the “prepare()” method is called to prepare the file, and then the start() method. There has to be permission for MIC to function.

5.4.4 Video Player

The video playback in Android is somewhat simplified for developers to implement with the provision of VideoView and MediaController, although it still requires more effort than the audio player. Videos can be played from the device SD card and from the Web server. Compared to other languages, Android SDK provides some additional

abstractions that take care of some complex issues in video applications. The “android.widget.VideoView” is a specialized view control that contains the creating and initializing of the MediaPlayer. If a VideoView is created and linked to the Activity, and the path to the video file is also set, calling the “start()” method prompts the video file to be played, provided such a file is supported by Android.



Figure 21. Video player screen diagram

The diagram in figure 21 shows the video player in a landscape view. The MediaController takes charge of the buttons shown on the screen. The buttons appear when the screen is touched and disappear after a few seconds. The application also uses the “OnCompletionListener()” method to check for the end of the video file. When the file finished playing the “finish()” method is called, this returns back to the Activity which holds the list of video files. Figure 22 below shows the code snippet of this application.

```

VideoView myVideoView;

myVideoView = (VideoView)findViewById(R.id.<id_in_the_xml_file>);
myVideoView.dispatchWindowFocusChanged(false);
MediaController mc = new MediaController(this);
myVideoView.setMediaController(mc);
mc.show();
myVideoView.requestFocus();
myVideoView.setVideoPath("file://" + <file_in_SDcard/web_server>);
myVideoView.start();
myVideoView.setOnCompletionListener(new
MediaPlayer.OnCompletionListener() {

    public void onCompletion(MediaPlayer myVideoView) {

        finish();
    }
}
);

```

Figure 22. Sample code for video player

The code in Figure 22 shows part of the application video player implementation, the “.xml” file in the layout directory holds the VideoView.

5.4.5 Audio Player

The audio player is implemented in a similar way as the video player except that it does not require the VideoView. As explained in chapter 3, the audio player uses the MediaPlayer class. If an instance of MediaPlayer is made, it will be used to call the “setDataSource()” method. This method sets the application to where the file is situated, (in this case, in the sdcard of the device). Then the “prepare()” method is called to prepare the media player for playback. The next method after that is the “start()” method which actually starts playing the file.

Figure 23 below shows the audio player of this project. It has a list of audio files to be played, a pause button, stop button and re-start button. Pressing directly on any file on the list automatically triggers the process to start playing the file. The stop button is implemented to call the “stop()” method while the file is playing. The pause button calls the “getCurrentPosition()” method. This method, as the name implies, gets the

current position of the player. The Re-start button is used to start the player from the pause mode, which implies that while the MediaPlayer instance is not equal to null and it is not playing, then the “start()” method is called. The current position obtained previously is passed to the “seekTo()” method, which makes it start playing from that particular position.

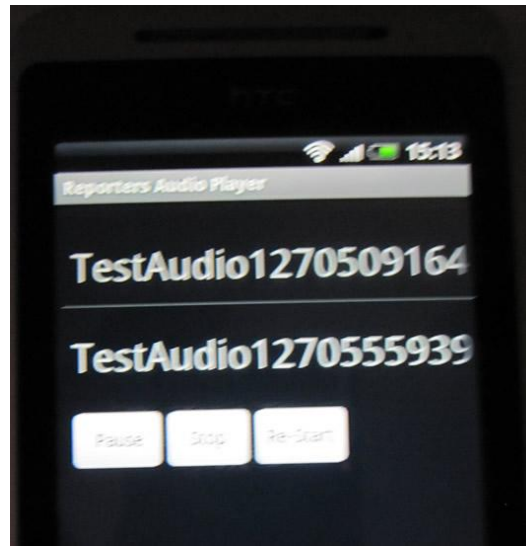


Figure 23. Audio player diagram

Figure 23 shows the screen of the application audio player, the test file can be played by click on the file.

5.4.6 Sending file to the server

This project focus more on how the video files can be sent along with other meta-data to the server. In this process the application is made to get the location of the device, the date and time at which each file was sent, the username of the sender and the category assigned to the video file. All these parameters are used to identify each file in the server side. The server side uses PHP script to handle the files and the related meta-data, because PHP scripts have an efficient way of taking care of server side applications. It also has the capabilities of binding other transcoding executable files, if needed.

Uploading process

The Activity extends “ListActivity” and implements “Runnable”, which implies that having parsed the meta-data to the Activity, a list of all the video files is shown. The application then runs another thread on the background which writes the selected file and its meta-data to a buffer and sends it to the server. At this point the system runs a progress bar which mimics the idea that the process is still going on, and most importantly, protects the screen from the user’s interruption of the process.

```

public void run() {
String FileName = MEDIA_PATH + useFileName;

URLConnection kornet = null;
DataOutputStream dos = null;
DataInputStream inStream = null;

String Endings = "\r\n";
String Hyphens = "--";
String boundaries = "*****";

URL url = new URL(path_to_server_script); //opening the connection
kornet=(URLConnection)url.openConnection();//open http
conection
kornet.setDoInput(true); //Allow inputs
kornet.setDoOutput(true); //Allow outputs
kornet.setUseCaches(false);//Can't use cached one
kornet.setChunkedStreamingMode(1024*1024);
kornet.setRequestMethod("POST"); //Post method
kornet.setRequestProperty("Connection", "Keep-Alive");
kornet.setRequestProperty("Content-Type", "multipart/form-data;
boundary="+boundaries);
dos = new DataOutputStream(kornet.getOutputStream() );

dos.writeBytes(Hyphens + boundaries + Endings);//one meta-data
dos.writeBytes("Content-Disposition: form-data; name= username;" +
Ending);
dos.writeBytes(Ending);
dos.writeBytes(theName);
dos.writeBytes(Ending);

//Other meta-data here, if need be.
//buffer here.

dos.writeBytes(lineEnd);
dos.writeBytes(twoHyphens + boundaries + Hyphens + Endings);
//close stream,
dos.flush();dos.close();
//Server respons here.
}

```

Figure 24. Connecting and uploading sample code

Figure 24 shows a code snippet for sending the file to the server, and the path to the server script. An http connection and its methods that give optimum solution to the file streaming are specified. For example the “konet.setChunkedStreamingMode(1024*1024)” was used since the content length is not known in advance, the post method was used for sending the form. This concept also allows input and output in the connection, so that the server script can report back to the application on the possible outcome on how the file was handled in the server side. It is made for the connection to stay alive throughout this process. A ” Content-type” header of a “multipart /form-data” was used to the get the “String” attributes and their values, such as filename, date and time, locations and descriptions. This header uses boundaries which start with two hyphens, to separate the various parts of the request. Writing all the properties to a buffer and closing all the contacts automatically sends the file and its metadata to the server side.

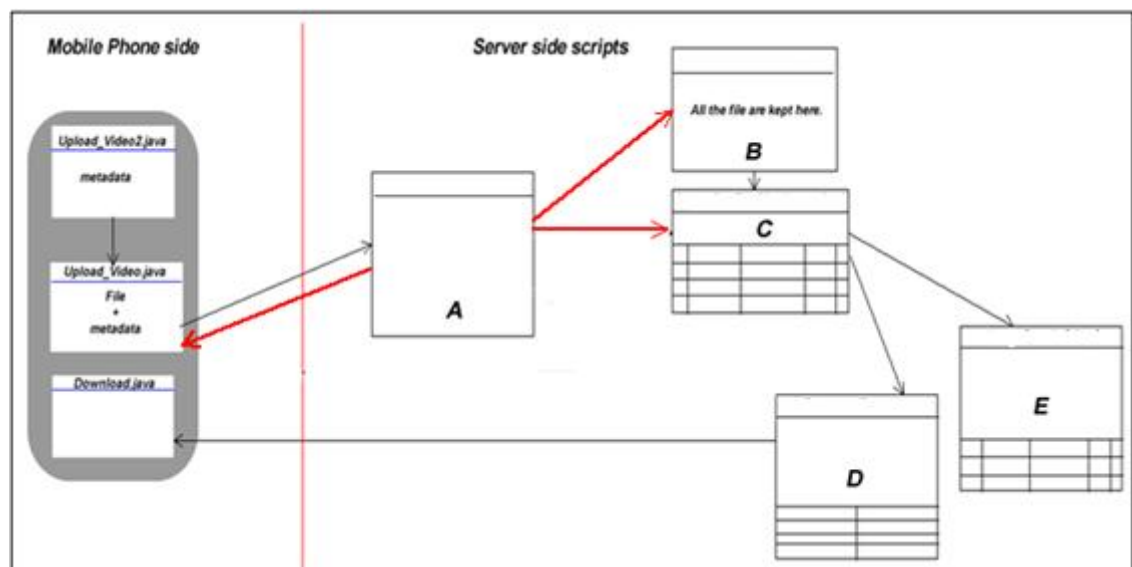


Figure 25. Data flow to the Server-side

The server side uses the PHP script to handle the request. Considering the diagram in figure 25, files A, D, E are PHP files while B is a directory in the server and C is a table in the database. When the file is sent, file A handles the sent video file and its metadata, the file is moved to the directory B, and details of the file and its metadata are registered in the database table, provided all the conditions are favorable. At this point

the phone receives the positive response of having uploaded successfully. If the process does not go as intended, maybe due to a connection failure, a filename already existing in the directory, or any possible cause that warrant the failure to upload the file to the server, the PHP script will send a response accordingly. File D and E rolls out all the files from the database table to the downloading phase and possible administrative uses respectively.

Downloading video files to the SDcard was implemented using the Android “WebViewClient” and java “FileOutputStream” functionalities. Downloading of the video file is made open for now, but in subsequent versions of the application, security measures will be considered before the file can be downloaded. The URL path to the files in the directory B are contained in the hyperlinks for each file name in file D, which implies that file D has the file names of all the files in directory B and also points to where they are kept in directory B. File D is a PHP file that rolls out all the file names and sizes from the database table. It appears on the phone using the Android WebView. Any file name that is pressed triggers the hyperlink to extract the file name and parse its URL path to the “FileOutputStream” object, to possibly obtain its byte data and store locally in the phone as a file. Appendix D shows the download view of the application.

Progress bar

In Android dialogs are used to interact with the user, which is helpful for notification purposes. A dialog is a window which appears in front of the current Activity, thereby causing the Activity to lose focus. A “ProgressDialog” is an extension of the “AlertDialog class”, which displays an animated spinning wheel or a progress bar. This gives the user an idea that an operation is taking place within the system.

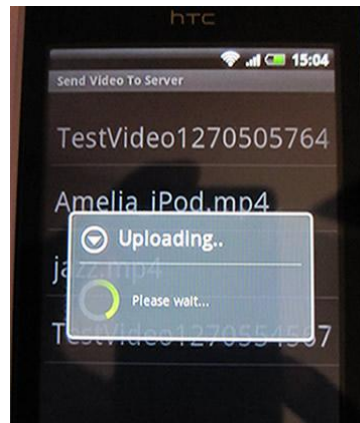


Figure 26. Application progress bar

The progress bar in figure 26 above was used in this project to notify the user that the file is on its way to the server. Thus the user cannot interrupt the process until a server response is back to the user, telling him or her if the uploading process is successful or not. Below is the code snippet in figure 27.

```
private ProgressDialog pdialog;
pdialog=ProgressDialog.show(Upload_Video.this,"Uploading..","Please wait...", true,false);
Thread thread = new Thread(this);
thread.start();
}
```

Figure 27: Sample code for progress bar.

Clicking on the file causes the progress dialog to appear, thereby starting another thread which sends the file to the server. The ProgressDialog is stopped by calling the “dismiss()” method. This depends on the feedback received from the server. A handler is created, and an integer value is parsed to its “sendMessage()” method, to differentiate the response sent from the server side.

```

try //Server Response
{
InputStream instrm = korret.getInputStream();
if( instrm == null ) {
throw new Exception( "Got a null content object!" );
}
StringBuffer putBackTogether = new StringBuffer();
Reader rid = new InputStreamReader( instrm, "UTF-8" );
char [] cbyte = new char [ 2048 ];
int volumRd = rid.read( cbyte );
while ( volumRd > 0 ) {
putBackTogether.append( cbyte, 0, volumRd );
volumRd = rid.read( cbyte );
}
String pageSource;
pageSource = putBackTogether.toString();
fidback=pageSource;
} catch (Exception e){
e.printStackTrace();
}
if(fidback==null){
handler.sendMessage(-1);
}else if(fidback.trim().contains("OK")){
handler.sendMessage(0);
}else if(fidback.trim().contains("alreadythere")){
handler.sendMessage(1);
}else {
handler.sendMessage(2);
}}

private Handler handler = new Handler() { //Handler
@Override
public void handleMessage(Message msg) {
pd.dismiss();
if(msg.what==-1){
AlertTwoBtn("No server reply!", "Do you wish to check for network?");
}else if(msg.what==0){
AlertOn("Success!", "File has been uploaded.");
}else if(msg.what==1){
AlertOn("Sorry!", "File name already exist.");
}else if(msg.what==2){
AlertOn("Error!", "File was not uploaded.");
}else{
AlertOn("Error!", "There is problem in uploading.");
}}};

```

Figure 28. Handling the Server side response

Figure 28 shows the code used for receiving and handling the server side response.

AlertDialog

The alert dialog is used here to notify the user of the feedback from the server side and for confirmation purposes. This class extends the “Dialog class”, and is constructed within the calling Activity. The alert dialog does not need to be registered in the manifest file. It uses the “setTitle()”, “setMessage()” and the “setButton()” methods to add the title, caption and buttons respectively on the alert window.

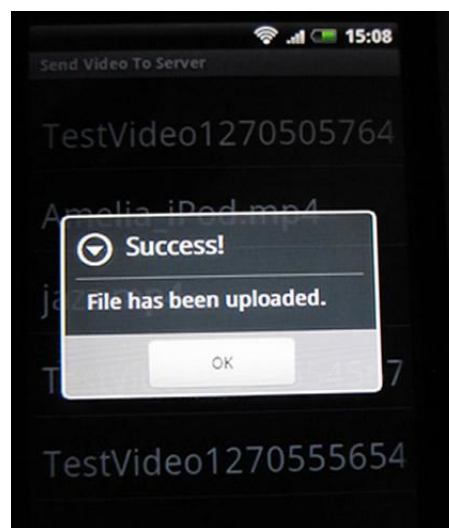


Figure 29. Application sample alert dialog

Figure 29 shows a sample of a successfully uploaded file to the server. The alert dialog was called by the progress bar handler, and a string variables was parsed to the “AlertOn()” methods. To make the alert dialog appear on the screen the “show()” method has to be called by the instance variable. The code snippet in figure 30 below shows the implementation.

```
private void AlertOn(String Title,String message) {
    AlertDialog theDialog = new AlertDialog.Builder(this).create();
    theDialog.setTitle(Title);
    theDialog.setMessage(message);
    theDialog.setButton("OK", new DialogInterface.OnClickListener() {
        public void onClick(DialogInterface dialog, int which) {
            return;
        }
    });
    theDialog.show();
}
```

Figure 30. Alert dialog code snippet

5.4.7 Search for WLAN

This part of the project was to help the user check if there is any wireless connection available in the area. It works by scanning for a wireless connection, by accessing the Wi-Fi Manager using the “getSystemService()” method and parsing the “Context.WIFI_SERVICE” object. The activity also contains a class that extends BroadcastReceiver, which gets a list of the scanned results, by calling the “alertWlan()” method if the size of the list is greater than one.

Being greater than one implies that there is at least one connection available. The “alertWlan()” method is implemented to start the “Settings.ACTION_WIFI_SETTINGS”, which brings the user to the point of connection to the wireless LAN. Figure 31 below shows the diagram of the interface.



Figure 31. Application WLAN connection sample

Figure 31 shows that there is at least one or more wireless hotspots in the area where the testing took place. The “OK” button takes the user to the connection point where further authentications will be made before connecting.

5.4.8 Deleting file

The application is also implemented such that unwanted video files can be deleted from the sdcard. Considering the code snippet in figure 32, the “onListItemClick()” method holds the position of the available file names in the list. When a file name is clicked on the list, the position is parsed to the “get()” method. Calling the instance of the array list on the “get()” method automatically points to the file in the sdcard. The “delete()” method is then used to delete the file.

```

@Override
public void onListItemClick(final ListView l, final View v, final int position, long id) {
    super.onListItemClick(l, v, position, id);
    AlertDialog deleteDialog = new AlertDialog.Builder(this).create();
    deleteDialog.setTitle("Deleting File !");
    deleteDialog.setMessage("File will delete permanently !");
    deleteDialog.setButton("OK", new DialogInterface.OnClickListener() {
        public void onClick(DialogInterface dialog, int which) {
            File file = new File(MEDIA_PATH + songs.get(position));
            file.delete();
            l.setEmptyView(v);
            return;
        }
    });
    deleteDialog.setButton2("Cancel", new DialogInterface.OnClickListener() {
        public void onClick(DialogInterface dialog, int which) {
            return;
        }
    });
    alertDialog.show();
}

```

Figure 32. Code snippet for deleting files

Figure 32 shows the implementation of the alert dialog which appears when a file is clicked for deleting.

5.4.9 Application Manifest File

Android Manifest file is an “.xml” file where the global settings of the given project are made. It lets one define the structure and metadata of an application and its components such as Activities, Content Providers, Services, Broadcast Receivers, Icons and Themes. Newly created activities have to be added to the manifest file for it to function as part of the project. A permission required for using the Internet, device camera, WiFi, and Record Audio, are declared in the “AndroidManifest.xml” file. Figure 33 shows the application manifest file.

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

    <application android:icon="@drawable/icon" android:label="Reporters Video Project">
        <activity android:name=".ReportersVideoProject" android:label="Reporters Video Project">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <activity android:name=".Reporters_Audio_Player" android:label="Reporters Audio Player">
        </activity>
        <activity android:name=".Reporters_Video_Player" android:label="Reporters Video Player">
        </activity>
        <activity android:name=".Upload_Video" android:label="Send Video To Server ">
        </activity>
        <activity android:name=".Upload_Video2" android:label="Send Video To Server ">
        </activity>
        <activity android:name=".Video_Service" android:label="Save Edit or Delete ">
        </activity>
        <activity android:name=".Video_Screen" android:label="Current Video "
            android:theme="@android:style/Theme.NoTitleBar">
        </activity>
        <activity android:name=".Record_Audio" android:label="Reporters voice Record">
        </activity>
        <activity android:name=".Record_Video" android:label="Reporters Video Record">
        </activity>
        <activity android:name=".Reporters_WLAN" android:label="Search for Wifi network">
        </activity>
        <activity android:name=".Delete_Video" android:label="Deleting Video File">
        </activity>
        <activity android:name=".Download" android:label="Download Video File(s)">
        </activity>
        <activity android:name=".Downloading" android:label="Press the button below">
        </activity>
    </application>
    <uses-sdk android:minSdkVersion="3" />
    <uses-permission android:name="android.permission.RECORD_AUDIO"></uses-permission>
    <uses-permission android:name="android.permission.CAMERA"></uses-permission>
    <uses-permission android:name="android.permission.INTERNET"></uses-permission>
    <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION"></uses-permission>
    <uses-permission android:name="android.permission.ACCESS_WIFI_STATE"></uses-permission>
    <uses-permission android:name="android.permission.CHANGE_WIFI_STATE"></uses-permission>
</manifest>
```

Figure 33. Diagram of the Manifest file

Figure 33 shows the application manifest file which binds the properties and components that constitute the application. The icon (which is kept in the “Resource Directory” of the application) is defined here alongside with the labels. Every activity used in this application is displayed within the “intent-filter” to avoid a runtime exception.

5.5 Feedback and Future Implementations

The application was evaluated by researchers from the Tampere University of Technology / The Unit of Human-Centered Technology and Managing Editor from Sanoma Kaupunkilehdet. The modifications are stated as follows:

- Changing the user interface to the Finnish Language: The entire user interface of the application was designed to be in the English language, but it has to be changed into Finnish language.
- Collection of phone number: The number will be asked and stored when the application is started for the first time. This will replace the previous user name that was asked.
- Creating metadata: The application records video and it collects metadata at the point of sending the file to the server, but it has to be modified to collect the metadata (phone number, coordinates from GPS, date/time and category) at the end of recording each video.
- Categories: The categories for the videos have to be increased from three (3) to fifteen (15).
- Downloading video: The prototype only creates the functionalities of downloading files from the dedicated server, but security measures have to be put in place.
- Audio Recording was not needed at this time, so it was removed from the application.

The modified version was then evaluated by the above mentioned representatives and they gave positive feedback on the general functionalities of the application.

6 Conclusion

Android-based mobile phones are becoming more interesting. They have an easy-to-implement nature which makes them convenient for developers to easily transform helpful ideas into interesting applications. Android is open source, which implies that the software had been debugged by several developers worldwide, which has led to bug-free software. It has good APIs to be used for applications.

The aim of this project was to create an easy-to-use application that would be capable of recording videos and send the video files along with other meta-data to the server side. The application created has several features such as video recording, video player, location-based mechanism, and wireless network functionalities. PHP was used as the server side script of the application with some configurations made in the server to accept larger amounts of file at a time.

This project was a prototype that can also be customized to the needs of freelancers, user-generated contents, citizen media and other media and media-related organizations. It also sets the basis for other developers who want to continue in this line in the future. The application was tested and it received a positive feedback from researchers from the Tampere University of Technology and journalists from Sanoma Oy.

Software development is prone to have additional implementations and modify existing ones as time goes by, and the Development of Reporter's Video Recorder Based on Android is not an exception. The application can be extended in many ways.

Functionalities such as taking photos can be added to it, and could be made so that photos could be taken alongside with metadata collection, deleting photos and its metadata, showing the photos that have been taken, sending photos to the server and also downloading photos from the server. All these could possibly be incorporated in the next release of the software (version 2.0).

The application presently supports a WLAN connection, it can be further improved to support a 3G connection. This will enhance the efficiency in the areas of coverage, faster downloading and uploading of files to the server. Also, defined rules have to be set in place on how to download videos or photos from the server.

The application has to be integrated into the Sanoma Oy's server. Since this work was done using the school server (Helsinki Metropolia University of Applied Sciences), it has to be moved to the company's server for operation to start off.

References

- [1] Wikipedia. The free encyclopedia [online]. Co-creation.
URL:<http://en.wikipedia.org/wiki/Co-creation>. Accessed 13 May 2010.
- [2] June Kaminski. Harnessing the wave of co-creation [online] 2009.
URL: http://ojni.org/13_3/june.pdf. Accessed 06 April 2010.
- [3] E-cocreation. University of California, Berkeley. Strategic Computing and Communications Technology. Business models of co-creation.
- [4] Wikipedia. The free encyclopedia [online]. User-generated content.
URL: http://en.wikipedia.org/wiki/User-generated_content. Accessed 27 June 2010.
- [5] Wikipedia. The free encyclopedia [online]. Citizen Journalism.
URL: http://en.wikipedia.org/wiki/Citizen_journalism#cite_note-glaser2006-4.
Accessed 28 June 2010.
- [6] Mark Glaser. MediaShift [online]. Your guide to citizen journalism.
URL: <http://www.pbs.org/mediashift/2006/09/your-guide-to-citizen-journalism270.html>. Accessed 28 June 2010.
- [7] Rick Rogers, John Lombardo. Android Application Development, 1st Edition 2009.
- [8] Android Developers. Download the Android SDK [online]. March 2009
URL: <http://developer.android.com/intl/de/sdk/index.html>. Accessed 12 February 2010.
- [9] Sample Code | Android Developers. Sample Code [online].
URL: <http://developer.android.com/intl/de/guide/samples/index.html>. Accessed 12 February 2010.
- [10] What is Android | Android Developers; Android Architecture [online].
URL: <http://developer.android.com/intl/de/guide/basics/what-is-android.html>.
Accessed 18 February 2010.
- [11] Nicolas Gramlich : Andbook, release .002.
- [12] Jerome(J.F) DiMarzio : A programmer's guide, 2008.
- [13] Reto Meier: Professional Android Application Development, 2009.
- [14] Android Competency Center. Android Application Lifecycle [online]. January 2009; URL: <http://www.androidcompetencycenter.com/2009/01/android-application-lifecycle/>. Accessed 4 March 2010.

- [15] Android Developers. Component Lifecycles [online].
URL: <http://developer.android.com/intl/de/guide/topics/fundamentals.html>. Accessed 5 March 2010.
- [16] Activity Manager. Open Source Java [online]. March 2006.
URL: <http://www.oreillyn.com/pub/d/1540>. Accessed 5 March 2010.
- [17] Android Developers. Activity Lifecycle [online].
URL: <http://developer.android.com/intl/de/reference/android/app/Activity.html>.
Accessed 8 March 2010.
- [18] Content Provider | Android Developers. Content Providers [online];
URL: <http://developer.android.com/intl/de/guide/topics/providers/content-providers.html>. Accessed 12 March 2010.
- [19] Chris Haseman: Android Essentials firstpress. 2008.
- [20] Sayed Y. Hashimi and Satya Komatineni: Pro Android. 2009.
- [21] W.Frank Ableson, Charlie Collins, Robi Sen: Unlocking Android 1.1, A Developer's Guide. 2008.
- [22] Android Developers. Audio and Video [online].
URL: <http://developer.android.com/intl/de/guide/topics/media/index.html>. Accessed 15 March 2010.
- [23] Android Developers. Class Overview [online].
URL: <http://developer.android.com/intl/de/reference/android/media/MediaRecorder.html>. Accessed 26 March 2010.
- [24] Mark L. Murphy: Beginning Android. 2009.
- [25] IT WIZARD. Broadcast Receiver [online]. August 2009.
URL: <http://www.itwizard.ro/broadcast-receiver-112.html>. Accessed 30 March 2010.
- [26] Wikipedia. The free encyclopedia [online]. YouTube.
URL: <http://en.wikipedia.org/wiki/YouTube>. Accessed 02 April 2010.
- [27] Developer's Guide. Data API Protocol-YouTube. Resumable uploads.
URL: http://code.google.com/apis/youtube/2.0/developers_guide_protocol.html#Direct_uploading. Accessed 05 April 2010.

Appendices

Appendix A: Application main view



Figure 34. Application desktop in portrait view

As soon as the application is launched, it appears like figure 33 above, this tells the user to press the menu and choose from the menu list.



Figure 35. Application desktop in landscape view

The phone changes to the landscape view automatically when the phone is turned horizontally.

Appendix A: Application main view

When the menu is pressed the view show the list of "Activities" that the system has, pressing on any of them takes the user to that particular activity.



Figure 36. Menu view (portrait)

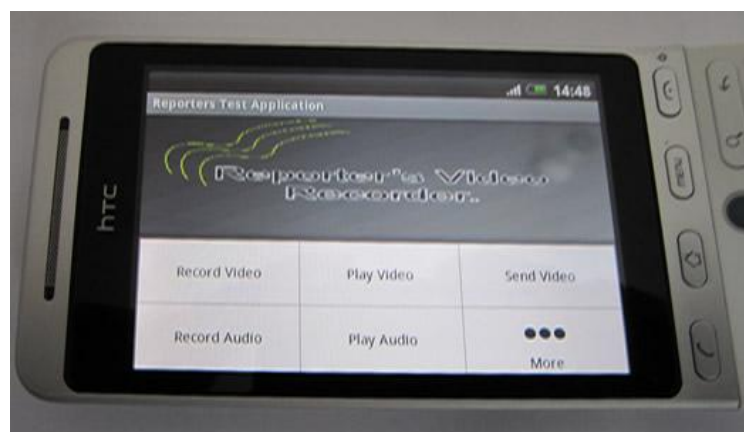


Figure 37. Menu view (landscape)

Appendix B: Video recording

This illustrates the video recording of the project, it gives a very quality video, The view is specifically set to be horizontal. When the stop button is pressed it automatically takes the user to another view where the default file name can be changed and the file could either be saved or deleted.



Figure 38. Testing the video recorder

Appendix B: Video recording

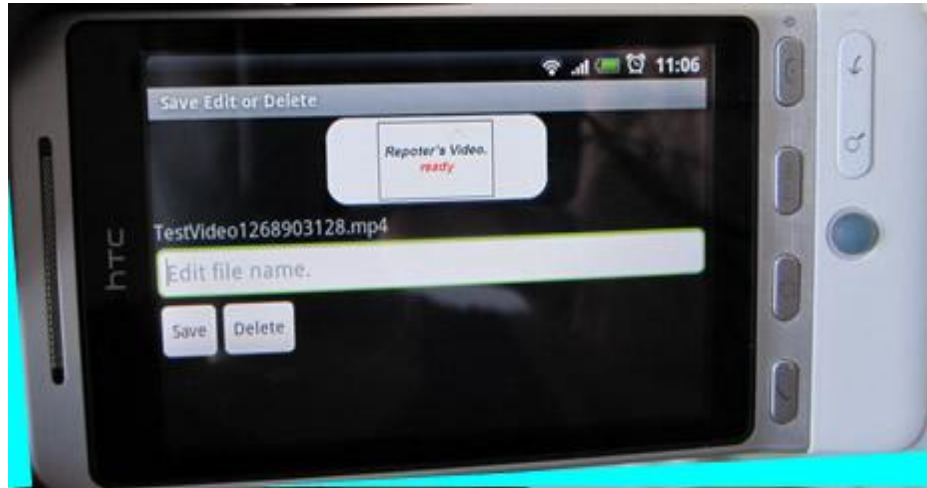


Figure 39. Saving the file name

Pressing the edit text view automatically brings out the key pad to type the file name and then save the file, or the user may even choose to delete the file at the point. After saving or deleting is done the application automatically take the user back to the video recording view again.

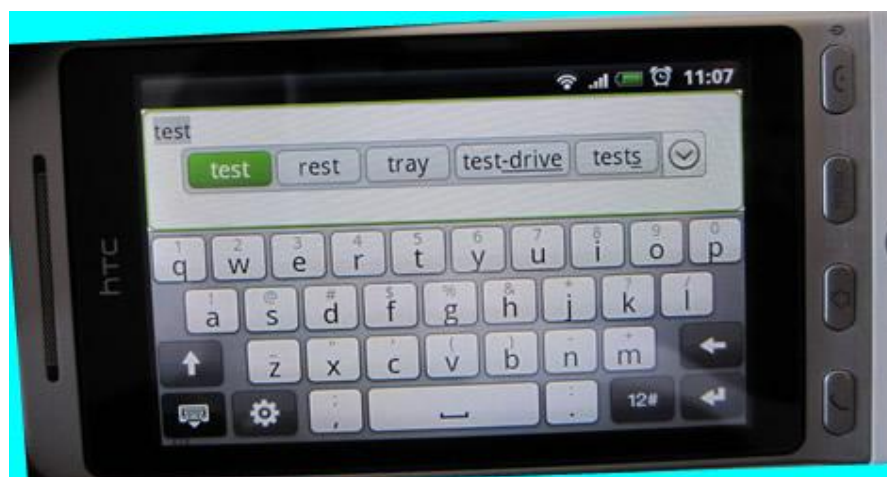


Figure 39b. Saving the file name

Appendix C: Uploading files

Provided there is a functional wireless connection available, files can be sent to the dedicated server efficiently, the application gets the address of the location, time and date automatically, the user has to enter the “User Name” and choose a category for the file to be sent.

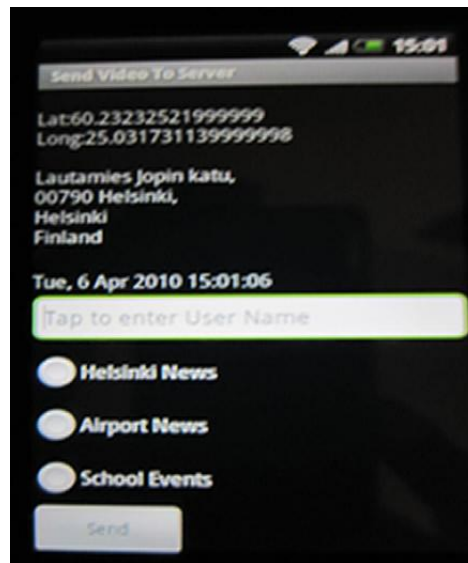


Figure 40. Collecting meta-data

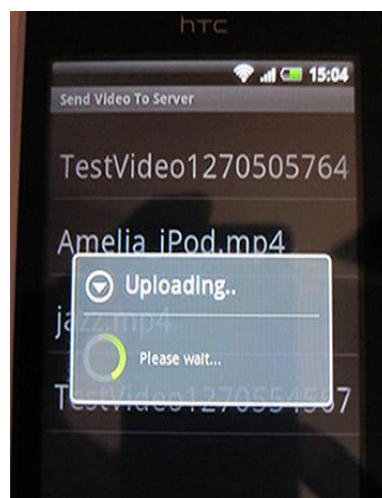


Figure 41. Uploading in progress

Appendix C: Uploading files

The system checks for WLAN within the area of the mobile phone. Figure 41 shows the list of available networks in the area where this test is made. The application functions very well when the username and the password of the wireless connection is entered correctly.

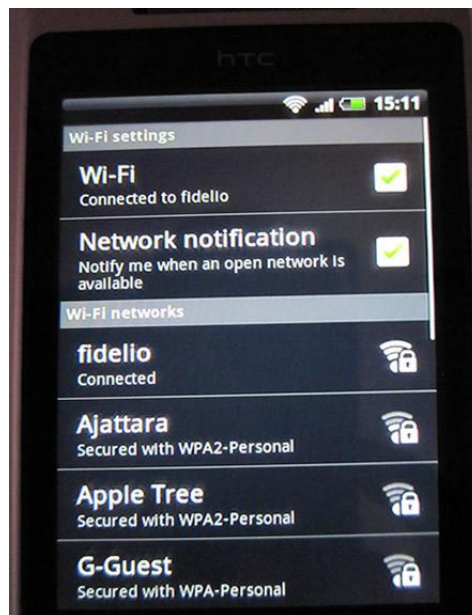


Figure 42. Checking for available wireless networks

Appendix D: Downloading video files



Figure 43. Downloading files

Figure 43 shows the view where files can be downloaded from the server to the SDcard. The view can be zoomed for clarity purposes. Pressing on any of the file names implies the intension to download such file, and this takes you to figure 44 below to carry on with the downloading process.

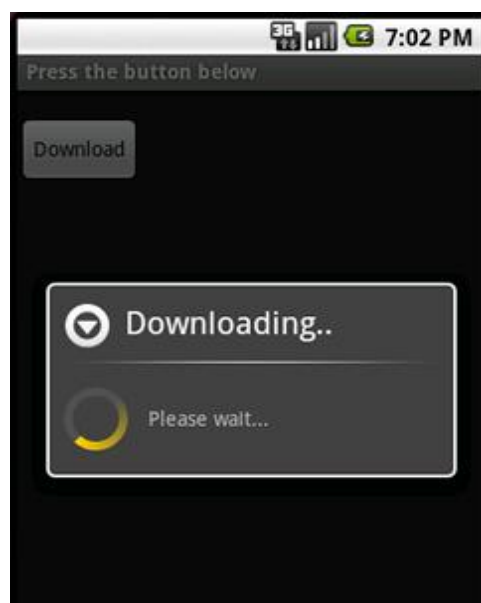


Figure 44. Downloading in progress

Appendix E: Playing Video



Figure 45. Video files

Videos plays automatically when the file name is pressed in figure 44, at the end of the file the system goes back to video file list.



Figure 46. Playing Video.