MAJOR PROJECT REPORT

ON

"STUDENT TRANSPORT TRACKING WITH NEAR FIELD COMMUNICATION (NFC)"

Submitted In Partial Fulfilment Of The Requirements
For The Award Of The Degree Of

Bachelor Of Technology
In
Computer Science And Engineering

Guide:

Ms. Sakshi Malhotra
Assistant Professor
Deptt Of Computer Science

Submitted By:

Harshit Trivedi (06496502710)

HMR INSTITUTE OF TECHNOLOGY & MANAGEMENT
HAMIDPUR, DELHI 110036

Affiliated to
GURU GOBIND SINGH INDRAPRASTHA UNIVERSITY
Sector - 16C Dwarka, Delhi - 110075, India

2010-14

CERTIFICATE OF ORIGINALITY

This is to certify that the Major Project Report entitled **"Student Transport Tracking With Near Field Communication (NFC)"** developed by HARSHIT TRIVEDI (06496502710) is an authentic work carried out by them under my guidance and supervision. The matter embodied in this project has not been submitted earlier for the award of any degree or diploma to the best of my knowledge and belief.

DATE:

SIGNATURE OF GUIDE:

Ms. Sakshi Malhotra
Assistant Professor
Deptt Of Computer Science

DECLARATION

I hereby declare that the Major Project entitled **"Student Transport Tracking With Near Field Communication (NFC)"** submitted to HMR Institute Of Technology And Management in partial fulfilment of requirement for the award of degree of Bachelor Of Technology in Computer Science And Engineering is a record of bonafide project work carried out under the guidance of Ms. Sakshi Malhotra.

I further declare that this project has not been submitted for the award of any other degree in any other institutes or universities.

DATE:

_____

Harshit Trivedi
(06496502710)

ACKNOWLEDGEMENT

I take this opportunity to express our profound gratitude and deep regards to my guide Ms. Sakshi Malhotra for her exemplary guidance, monitoring and constant encouragement throughout the course of this project.

I also take this opportunity to express a deep sense of gratitude to the faculty of Computer Science Department for their cordial support and guidance, which helped us in completing this task through various stages.

Lastly, we thank the almighty, our parents and friends for their constant love, support and encouragement without which this project would not be possible.

Sincerely,

Harshit Trivedi
h.trivedi04@gmail.com
(06496502710)

ABSTRACT

Smartphones are no longer just fancy mobile devices that let you e-mail and surf the Web. A contemporary smartphone has more computing power than all of the computers that were at NASA's disposal back in 1969 when the United States first landed on the moon. Although you probably won't use your phone to control your own lunar lander anytime soon, it is likely to replace all sorts of other nifty stuff you use daily such as credit cards, car keys, tickets, health cards, and hotel room access cards because NFC (Near Field Communication)-enabled mobile phones will provide all these functionalities.

NFC (near field communication) is a wireless technology which allows for the transfer of data such as text or numbers between two NFC enabled devices, typically requiring a distance of 4cm or less to initiate a connection. NFC allows you to share small payloads of data between an NFC tag and an Android-powered device, or between two Android-powered devices. NFC tags, for example stickers or wristbands, contain small microchips with little aerials which can store a small amount of information for transfer to another NFC device, such as a mobile phone.

'Student Transport Tracking With NFC' is a smart NFC integrated application for Android powered smartphones that enables any institution to track the students' transportation activity from home to school and from school to home.

The application simply records and saves the getting-on and getting-off time of every student on the bus, which is done by reading the NFC tags (which can be modified to replace a student's school ID card) through an NFC-enabled Android smartphone present with the driver. It has the ability to write and register a new ID tag for a new student too. The application also has the ability to send the day's transport tracking record to the Dean, the Transport Incharge or the parents of the students.

In this way, an institution can monitor and track every student's transportation record, and at the same time parents can be relaxed and untroubled about their child's whereabouts.

Armed with these tiny chips, smartphones are about to graduate from smart to downright brainiac status. NFC is certainly a technology to keep our eyes on as it has the potential to bring a new level of convenience to many aspects of a typical person's daily life.

PROBLEM STATEMENT

The aim of this project is to develop an Android based application that integrates all basic NFC tag functionalities in a single application interface to ease the user's NFC experience.

We intend to make an application that enables tracking the students' transportation activity from home to school and from school to home. In order to run the application, each student needs to have an NFC tag that will be encoded to contain the ID of that student. It is assumed that, inside the school bus, there is an NFC-enabled mobile phone, possibly owned by the driver that reads the tags of the students. When a student gets onto the bus, they touch their tag to the mobile and the mobile saves the getting-on time; when that student gets off from the bus, they again touch their tag to the mobile and this time the mobile saves the getting-off time.

The application also brings to fore the ability to send the day's getting-on and getting-off times to the Dean, the Transport Incharge or the parents of the students, so that the institution and the students' parents can keep track of their ward's activities and prevent any untoward doings.

TABLE OF CONTENTS

S.No.		Content	Page No.
i		**CERTIFICATE OF ORIGINALITY**	ii
ii		**DECLARATION**	iii
iii		**ACKNOWLEDGEMENT**	iv
iv		**ABSTRACT**	v
v		**PROBLEM STATEMENT**	vi
1		**INTRODUCTION**	11
	1.1	What Is Near Field Communication?	12
	1.2	How Does It Work?	12
	1.3	What Will We Use It For?	14
	1.4	Will It Catch On?	15
	1.5	Who Is Backing It?	17
2		**OBJECTIVES OF THE PROJECT**	18
3		**LITERATURE REVIEW**	19
	3.1	Introduction To Near Field Communication (NFC) Technology	19
		3.1.1 Energy And Data Transfer	20
		3.1.2 The NFC Protocol Stack	21
		3.1.3 More About NDEF	24
	3.2	Android Development	26
		3.2.1 What Is Android?	26
		3.2.2 Android Runtime	27
		3.2.3 Libraries	27
		3.2.4 Application Framework	28
		3.2.5 Applications	28
		3.2.6 Android SDK	28
		3.2.7 What You Need To Start?	28
4		**SELECTED SOFTWARE**	29
	4.1	Eclipse Standard IDE 4.3.2 (Kepler)	29
		4.1.1 Open Architecture	29
		4.1.2 Platform Structure	30
	4.2	Android SDK	32
	4.3	Android Development Tools	33
5		**SYSTEM DESIGN**	34
	5.1	Minimal Requirements	34
	5.2	System Requirements	35
6		**UML DIAGRAMS**	36
	6.1	Use-Case Diagrams	36

	6.2	Workflow Diagrams	40
		6.1.1 TransportationActivity	40
		6.1.2 TransportationWriterActivity	40
		6.1.3 WebServiceActivity	41
	6.3	Data Flow Diagrams	42
		6.3.1 Level-0 DFD	42
		6.3.2 Level-1 DFD	43
7		**TESTING**	44
8		**CONCLUSION**	46
	8.1	Some Final Thoughts	46
	8.2	Future Scope And Improvements	46
9		**APPENDIX**	47
10		**BIBLIOGRAPHY**	58

LIST OF ILLUSTRATIONS

Fig No.	Illustrations	Page No.
1	Components Of An NFC Tag	13
2	Uses Of NFC	14
3	NFC Types And Technology Types	21
4	NFC Protocol Stack	22
5	Relationship between NFC Stack Protocols and Android API	23
6	Core Libraries Of Android	27
7	Structure Of Eclipse Platform	30
8	Use – Case Diagram For Student Transport Tracking	36
9	Workflow Diagram For TransportationActivity	40
10	Workflow Diagram For TransportationWriterActivity	40
11	Workflow Diagram For WebServiceActivity	41
12	Level-0 DFD	42
13	Level-1 DFD	43
14	Home Screen Of 'NFC Student TransTrack'	47
15	Launching of TransportWriterActivity on tapping 'Write Student ID Tag'	48
16	'Touch NFC Tag to write' prompt on tapping 'Save To Tag'	49
17	'Message is written to tag' toast on succesful write operation	50
18	Launching of TransportationActivity on tapping 'Start Student TransTrack…'	51
19	Adding of student getting-on information on reading NFC Tag for the 1st time	52
20	Adding of student getting-off information on reading NFC Tag for the 2nd time	53
21	Deleted all items from memory on tapping 'Empty List'	54
22	Launching of WebServiceActivity on tapping 'Send Mail…'	55
23	Activity Launcher to choose application for sending the e-mail	56
24	Auto-synthesized mail prepared on tapping 'Send…'	57

LIST OF TABLES

Table No.	Table Name	Page No.
1	Major Runtime Components Of Eclipse	31
2	Use Case Descriptions	37
3	Test Cases	44

1. INTRODUCTION

Near Field Communication, or simply NFC, is shaping up to be one of the hottest tech trends of the next few years. Mobile payment systems backed by major financial institutions are either already being tested or in plans to start tests, while smartphones with built-in NFC chips are making their way into the U.S. and Europe. But beyond payments, NFC has the potential to reach many other industries, from location-based services to ticketing and public transportation.

It's not too far-fetched to imagine a world where all we need to carry around with us is a single do-it-all device. NFC could allow our smartphone to pay for products, open doors, as well as act as our personal ID or a virtual ticket for transport and attractions. Say goodbye to your keys, wallet, cards and any extra weight in your pockets.

Of course, there are loose ends that will need to be worked out before the technology actually catches on, like ensuring the proper infrastructure is there and addressing any security concerns. In this piece we are going to tell you the things that you should know about Near Field Communication and how it could make your life easier in the future.

1.1 What Is Near Field Communication?

Let's start with a basic definition:

Near Field Communication (or **NFC** for short), is a set of standards for smartphones and similar devices to establish radio communication with each other by touching them together or bringing them into proximity, usually no more than a few inches.

NFC is a wireless technology that makes use of interacting electromagnetic radio fields to transmit small bits of information between an "initiator" and a "target" -- a key card and your hotel room door, for example. The main idea behind NFC is to integrate wireless payment and tag reading in mobile phones along with peer-to-peer communication. It's similar to Bluetooth in the sense, that both are short-range communication technologies, and is considered a subset of existing RFID (Radio Frequency ID) standards given that is uses radio waves for identification purposes operating at 13.56 MHz. NFC devices can not only be used with the upcoming wireless payment terminals but can also replace contactless plastic cards used in the already established RFID infrastructure. But NFC has its unique set of characteristics that will determine how it's used in real-world practical applications.

For one thing NFC transmits data across much smaller distances, typically between 4 and 10 centimeters, compared to Bluetooth's 10-meter range, or in the case of some RFID implementations even kilometers. This by-design limitation reduces the likelihood of unwanted interception and makes NFC particularly suitable for crowded areas where correlating a signal with its transmitting physical device becomes difficult.

NFC technology development was initiated by Sony and Philips. The key feature that differentiates NFC from RFID is the possibility of bidirectional transfer of information which allows bidirectional communication between NFC devices. To connect two devices together, one simply brings them very close together or makes them touch physically.

Its short-range nature may significantly reduce the risk of eavesdropping but that alone does not guarantee secure communications; applications have to use higher-layer cryptographic protocols like SSL to establish a secure channel.

Another differentiating factor is that NFC sets up connections faster than standard Bluetooth and its low-power variant, Bluetooth 3.0. Instead of performing manual configurations to identify devices, the connection between two NFC devices is automatically established quickly in less than a tenth of a second. In fact, NFC could even be used to speed up the process of pairing two Bluetooth devices, acting as an initiator by simply bringing them close to each other.

Lastly, their data throughput capacity makes them fit for different applications. NFC operates within the globally available and unlicensed radio frequency ISM band of 13.56 MHz and can go up to a maximum data rate of 424Kb/s, whereas Bluetooth operates in the 2.4GHz frequency and can reach maximum data rate of 2.1Mb/s.

1.2 How Does It Work?

As mentioned before, NFC involves an initiator and a target, where the initiator *actively* generates an RF field that can power a *passive* target without an electricity source, meaning only one of the devices 'needs' to be powered. This enables NFC targets to take very simple form factors such as tags, stickers, key fobs, or cards that do not require batteries. Passive tags can therefore be much smaller and cheaper than active tags, which have their own batteries, although the reading range is more limited.

A simple example would be holding a NFC-equipped smartphone near a tagged movie poster and getting all relevant information in seconds -- trailer, reviews, schedules at the nearest theater and the option to buy tickets online. The smartphone would be the initiator and the tagged poster would be the passive target.

While the poster can only be a passive target, NFC-equipped devices like smartphones can act as initiators, targets, or both combined in an active peer-to-peer mode. Elaborating on the same

example, say you purchased a ticket to that movie from the poster, now you can bypass the line at the box office and redeem the ticket on your handset. The NFC reader at the movies is the initiator and reads the information from your phone, which acts as the target.

So, despite all the hype about disruption, NFC is more about increasing convenience than enabling something completely new. Instead of swiping a credit card or scanning a barcode just tap the NFC reader with your phone and off you go.

Android-powered devices with NFC simultaneously support three main modes of operation:

1. **Reader/writer mode**, allowing the NFC device to read and/or write passive NFC tags and stickers.

2. **P2P mode**, allowing the NFC device to exchange data with other NFC peers; this operation mode is used by Android Beam.

3. **Card emulation mode**, allowing the NFC device itself to act as an NFC card. The emulated NFC card can then be accessed by an external NFC reader, such as an NFC point-of-sale terminal.

NFC tags, for example stickers or wristbands, contain small microchips with little aerials which can store a small amount of information for transfer to another NFC device, such as a mobile phone. They both have a bit of storage memory, along with a radio chip attached to an antenna.

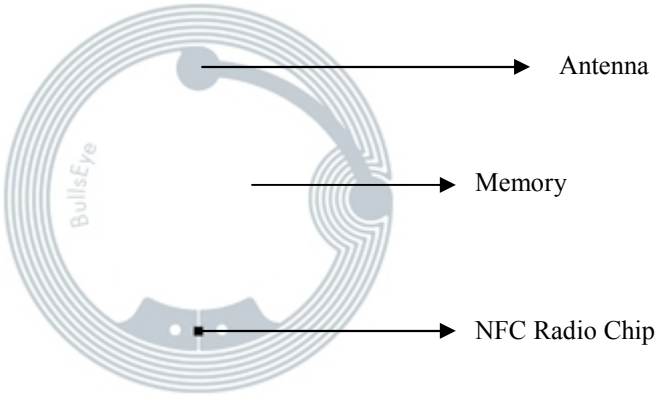


Figure 1: Components Of An NFC Tag

Tags can range in complexity. Simple tags offer just read and write semantics, sometimes with one-time programmable areas to make the card read-only. More complex tags offer math operations, and have cryptographic hardware to authenticate access to a sector. The most sophisticated tags contain operating environments, allowing complex interactions with code executing on the tag. The data stored in the tag can also be written in a variety of formats, but

many of the Android framework APIs are based around a NFC Forum standard called NDEF (NFC Data Exchange Format).

There are four flavors of NFC tags, types 1 through type 4, all featuring different capacities and data transfer speeds. **Type 1** tags typically store 96 bytes and work at 106 Kbps (kilobits per second); **Type 4**, the biggest and fastest, store up to 32 KB and work at speeds of up to 424 Kbps.

Type 1 and **Type 2** tags can be written to multiple times. These tags can also be permanently locked, or encrypted, so that no one can manipulate the data. **Type 3** and **Type 4** tags can only be written to once, like a CD or a DVD, and they lack the security of types 1 and 2.

1.3 What Will We Use It For?

Much noise has been focused on NFC's ability to power mobile payments and that's perhaps because the promise is enticing: It will turn your mobile phone into a wallet. Wave your NFC-equipped phone at a store reader and be on your way. NFC has the potential to replace your credit cards, checkbooks and other clumsy payment methods for a single device that you already carry everywhere. That sounds great if you ask me. So what are other possible uses?

Figure 2: Uses Of NFC

- **Public transportation.** This could arguably be a subset of mobile payments but it's worth mentioning on its own. In fact, in urban areas with high population density and good public transportation this can be a major driver of NFC adoption. Pilot and commercial programs have already been deployed in many cities of the world -- including my current city Nice, France -- where you can pay the bus, metro or tram with a tap of your phone.

- **Ticketing.** Take the movie theater example in the previous section and apply it to any kind of ticket: concerts or live shows, conferences, sporting events, theme parks, checking into a flight and boarding.

- **Keys.** Imagine getting rid of that extra weight in your pocket by replacing your entire keychain with your mobile phone. With a NFC-enabled phone you could potentially tap your way into your apartment, office or hotel room, start your car's engine, and access anything else that requires a key with one single device.

- **Comparison-shopping.** Whether you are doing groceries, buying clothes or getting something from the local electronics store, with a wave of your phone you could have access to reviews, additional product information, or prices from other stores. Much like you can do today with barcode scanning but faster.

- **"Check-ins" and venue reviews.** Google recently began pushing this by putting NFC-enabled Places stickers just outside some restaurants and businesses in the Portland, OR area. With your NFC-equipped phone you can easily rate places or read reviews so you have an idea if the food or service is any good before going in. It's also useful for getting 'point of interest' information in cities or location-based social networks like Foursquare.

These are just some of the practical applications of the technology. Google implementing NFC support natively within Android is a great boost for developers who want to create applications that rely on exchanging information based on proximity, and with smartphone makers starting to include NFC chips in their products we'll probably see numerous other examples of how we can make our daily interactions quick, effortless and more engaging.

The hallmark of NFC is ***'Instant Gratification'***, which means that no hassles with the connection setup or pairing of devices. Furthermore, it also provides a great experience for the user by launching directly into the application that handles the data that they are interacting with.

1.4 Will It Catch On?

We've been hearing about NFC and contact-less mobile payments in the U.S. for years, but it's only been during the past few months that there's been a rush of activity signalling that the technology might finally take off.

One issue is consumer behaviour but in many cases that can be fairly easy to overcome. In the case of mobile payments specifically there is bound to be some resistance over security concerns and it's up to payment processors to offer the appropriate safeguards. With a few exceptions around the world, most systems are still in trial, but looking at current implementations like MasterCard's PayPass you can get an idea of where things are going: zero liability protection programs, use of encryption technologies and requiring PIN codes for transactions over a certain amount.

The bigger issue is that NFC is not on the majority of phones and NFC readers are not widely available at merchants. This is slowly starting to change but it's not something that will happen overnight, and of course there comes the chicken-and-egg paradox: until more merchants accept contact-less payments there will be limited interest from customers in getting NFC-equipped phones or from manufacturers in selling them, and until you have a significant installed user base of consumer NFC devices, merchants won't be willing to invest in the payment-processing devices.

Big and small players are looking to make money through mobile payments and businesses are starting to take an interest in NFC as a way to improve engagement with customers at points of sale. I think there's a big chance NFC takes off in the next couple of years and we're getting front row seats to see who can make their solutions stick.

It's important to note that NFC or NFC-like systems are already widely used in some Asian countries, most notably Japan, which rolled out the first phones with built-in mobile payment chips as far back as July 2004. Today, about half of the country's more than 100 million subscribers have this capability, though some estimates put regular use at just 10%. The country uses a proprietary technology called FeliCa, developed by Sony, but eventually plans to move to the standardized NFC to avoid the higher licensing fees and increased cost of phones and readers.

In the U.S., Europe and several countries throughout the world there are numerous tests and pilot programs being conducted. While conducting research for this article I was hoping to play around with France's NFC implementation in Nice, dubbed Cityzi, but unfortunately several days after activating the service and despite many calls to get it sorted out, it is still not working. It's not clear to me if this is an isolated event but I'm sure many would turn away from the

technology if this was their first impression, which is why implementation and execution is so important for adoption.

Nevertheless, this pilot program that's been in effect since mid-2010, has been successful enough that is expanding to 9 more French cities and will be made available in a range of last generation handsets from key providers this year.

1.5 Who Is Backing It?

Mobile phone carriers, credit card companies, banks, companies like Google and PayPal, device manufacturers... there are financial gains in store for players across the field and with the hype surrounding NFC growing nobody wants to be left out. In the last few months almost every major mobile carrier in the U.S., banks and credit card companies have either announced plans or have started trialling NFC based payment systems.

PayPal, Google Checkout and Amazon Payments are also said to be working on NFC solutions, while phone makers and point-of-sale terminal vendors are including support for the technology in their latest devices.

2. OBJECTIVES OF THE PROJECT

The main objectives of this project are -

- To give an overview of the working and functionality of NFC technology
- To provide a comprehensive NFC integrated Android interface to the user to read and write data to NFC tags
- To develop an NFC-enabled system that enables and helps an institution to track the students' transportation activity between home and school by recording their getting-on and getting-off times

3. LITERATURE REVIEW

3.1 Introduction To Near Field Communication (NFC) Technology

NFC (Near Field Communication) is a wireless communication for devices in **very close range**, normally in physical contact. One of the two communicating devices may be **passive**, meaning that it has no energy source (no batteries, no solar cell).

The ancient relative of NFC is the anti-theft tag or security tag that is found in many stores. The door sensor generates a magnetic field. The tag has no batteries but can absorb field energy. This absorption is sensed and it triggers the alarm.

On top of this principle, there is the RFID technology. A RFID tag not only absorbs energy but also sends some information back. The "canonical" application of RFID is replacement or complement to barcode labels, since RFIDs can be read more easily. It creates a myriad of privacy problems e.g. you buy a T-shirt with your credit card, it has a RFID tag; you just become a person that is very easy to trace.

NFC goes one step further; the passive NFC device may be "intelligent", meaning that it possesses a tiny computer and can do some processing. The NFC potential is enormous and includes things like:

- Contactless credit cards. Making payments without taking the card out of the wallet.
- Public transport fare systems, tickets in general.
- Bonding and establishment of trust relationships, like Bluetooth pairing, without the need of PIN codes since the close contact is the security factor.
- E-money stored in your cell phone, use your phone as the credit card or pre-paid card. Many cell phones already support NFC.
- Communication with medical devices in a very ergonomic fashion, since the act of touching e.g. a blood pressure monitor with the cell phone is a strong indication that the user wants to transfer measurements. The proximity adds to security.

NFC allows communication between two "active" devices, e.g. two cell phones. This feature makes it easy to exchange contact cards, photos etc. The security and convenience are very nice: the interested parties just need to touch their gadgets. This still does not work as well as it should, because the standards are still emerging and there are incompatibilities between dissimilar platforms e.g. Windows Phone and Android.

3.1.1 Energy And Data Transfer

In NFC jargon, the passive device (that has no source of electricity) is simply named a **tag**, or sometimes a "card" since it takes the form factor of a credit card in most cases. The active device is called a **reader** because it typically just reads tags.

The reader and the tag have coils, not antennas. When the reader coil is energized, it generates a magnetic field. When a tag gets very close, the two coils form a transformer, and the tag can draw energy from the reader.

There is an apocryphal story about a ghetto in which people could illuminate their houses by drawing energy from an AM radio station nearby. This was discovered because the signal range was getting lower and lower. I don't know if there is any truth in this story, but it illustrates well the difference between *near field* and *far field*.

Far field is the electromagnetic radio signal that propagates for long distances. If I have a radio tuned on at home, this won't affect my neighbor's radio at all. Regardless of how many radios are receiving, the load on transmitter does not change; the parts are uncoupled.

In the other hand, the near field is composed of electric and magnetic fields that still did not combine into photons. Separately, they don't go very far. As the name suggests, the near field stays close to the antenna. A receptor placed in this region can absorb a lot of energy and hurt the radio's range. The transmitter can "feel" the presence of near-field receivers because they affect antenna's impedance.

The AM wavelength is around 500 meters, so the mythical ghetto would have to be within 80 meters of the AM station.

NFC puts these effects into good use to limit the reach and feed the passive tag. Since a coil is a bad antenna, little energy is wasted as radiation.

Since the tag has no power source, it communicates with the reader by **modulating its own energy consumption**. A tag that is more absorbent will force the reader to spend more energy. The reader monitors its own energy spending and decodes the information that the tag is "transmitting".

When two active NFC devices e.g. two phones are to communicate, there are some complications because both can try to send data at the same time (collision). The protocol must detect and resolve such collisions.

3.1.2 The NFC Protocol Stack

For the user, NFC is perfect. For the developer, NFC can be more complicated than initially expected. This has "historical" roots: the NFC specifications inherited pre-existing and incompatible flavours of NFC (from Philips, from Sony, and so on). The result is a stack with many redundant features, and an annoying multiplicity of tag types.

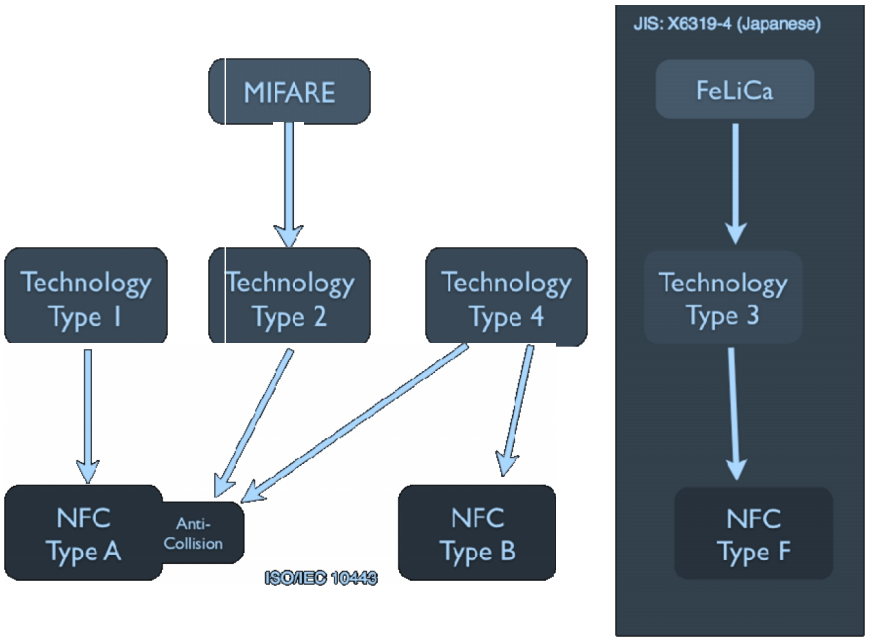


Figure 3: NFC Types And Technology Types

There are many tag "types": Type A, Type B, Type F... as shown in Figure 1. On top of these types, we have the "technology types": Type 1, 2 e and 4. For example, "Type 1" adopts a subset of the features implemented by "Type A".

Each type has a set of capabilities and limitations, e.g. tag chip memory size, maximum size of exchanged messages and so on. By the way, the NFC Forum allows free access to all specs, you just need to identify yourself before downloading.

Commercial standards like *MiFARE* implement additional features, like tag-embedded cryptography, on top of a basic type.

The good news is that NFC-capable phones like Android and Windows Phone can handle every tag type mentioned in Figure 3.

The NFC protocol stack makes the venerable TCP/IP look like child's play. See Figure 4.

Each tag type offers a different link protocol, and each technology type extends the command list of the base protocol. For example, NFC-A offers a command set, and Type 2 extends that set (kind of an heresy for people that believe in layered protocol stacks).

Because of that, there are many link protocols (the square hooks in Figure 2), and the transport/application protocols can hook to any link protocol. Because of that, I could not put all protocols in a layered picture, like textbooks do.

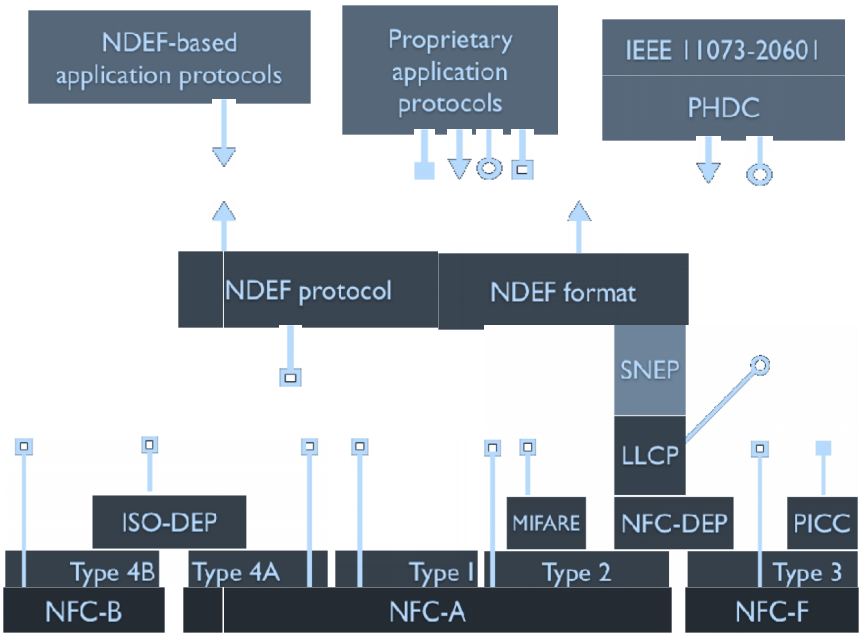


Figure 4: NFC Protocol Stack

Instead, I used "hooks" of different shapes to indicate who can connect with whom. For example, the NDEF transport protocol can employ any "open square" as link protocol, while the PHDC application protocol can use either NDEF ("arrow") or LLCP ("circle") as transport.

Fortunately for us developers, NDEF (NFC Data Exchange Format) is the most often used standard. NDEF specifies the message format stored in tag, as well as the access protocol that reads from and writes to the tag.

When the developer deals with NDEF, it is freed from dealing with the multiplicity of links and types, since normally the NDEF API hides the low-level details.

Since Android is the platform that stressed NFC support since the beginning, we will look how the Android API supports the many elements of the NFC protocol stack. I tried to reuse the "hooks" in Figure 5:

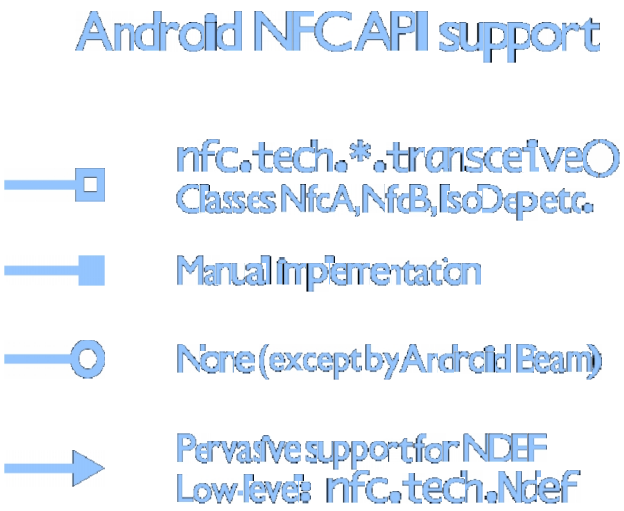


Figure 5: Relationship between NFC Stack Protocols and Android API

NDEF support is excellent, including tag writing support for any tag type.

Thanks to the "Intents" mechanism, an application can be configured to deal with NDEF messages of some given type. When such a tag is brought in contact, the app is loaded automatically. Android also offers a complete API for MiFARE standard.

The API offers the other extreme as well: raw access to link layers. This is a way to handle proprietary protocols, not based on NDEF. But the difficulty level is high since the developer will have to craft every packet from the link layer up, with all bits in their due places... and link layer packets do have different formats for each technology type.

Android also supports the ISO-DEP protocol, a link layer of a slightly higher level that abstracts Types 4A and 4B. The PICC protocol (from FeLiCa stack) can be supported through manual implementation, using the raw API for NFC-F.

There is no access to NFC-DEP protocol, which abstract Types 2 and 3. Likewise, there is no support do LLCP -- a transport protocol that borrows concepts from Ethernet (802.3 standard) including the MAC address concept.

LLCP offers connectionless and connection-oriented services, analogous to UDP and TCP. Its interface looks much more familiar to developers that have already worked with BSD Sockets, TCP/IP and Bluetooth. LLCP aims peer communication between active NFC devices e.g. two cell phones. (Communication with passive tags tends to be very short: read one NDEF message and disconnect.)

Android does not grant access to LLCP level via API, but it does use it internally. SNEP (Simple NDEF Exchange Protocol) is the vehicle of "Android Beam" feature and uses LLCP as transport. Curiously, the payload of a SNEP communication is simply a NDEF message. Unfortunately, is not possible to establish a two-way, continuous communication channel using the SNEP API.

Since part of my current work is related to medical devices, I have added the PHDC protocol to Figure 2. PHDC has the same role as USB's PHDC class and HDP profile in Bluetooth: to carry medical data that follows the IEEE 11073-20601 standard. PHDC can employ LLCP and NDEF as transports. Since NDEF message exchanges do not establish a connection by themselves, PHDC prescribes a "telegraphy" of messages to be sent back an forth, several per second.

Normally, a medical device will have batteries and/or will be connected to mains, but NDEF transport leaves the gate open for 100% passive devices e.g. a thermometer that only functions when activated by NFC reader.

Finally, there are the "cursed" proprietary protocols, conceived directly on top of NFC link protocols. I mourn the fellow programmers that have to deal with them.

3.1.3 More About NDEF

A **NDEF message** is formed by one or more **NDEF records**. Each record is a structure with many fields:

- MB bit (message begin)
- ME bit (message end)
- CF bit (chunk)
- Meta-type of the record (URI, MIME, RTD, etc.)
- Record type, format accordingly to meta-type
- ID (of variable size)
- Message payload

The existence of all these fields in each record solves a number of problems. A message can be sent in fragments (using MB and ME bits to fence the message boundaries). Messages of different types can be send through the same channel and easily separated by receiver. Different messages can be sent at the same time, using interleaved records, as long as each message have a unique ID.

The meta-type URI leaves the barn door open for domain-specific and application-specific types. For example, I can define a type like "urn:nfc:ext:epx.com.br:boo", that is employed only by my own applications. Naturally, each type imposes a inner structure on the message payload.

The NDEF protocol has two basic operations: read and write messages. They are built on technology-layer operations. The normative document of each tag type (Type 1, 2, 3 and 4) specifies how a NDEF shall be read from, written to and stored in the tag.

The NDEF operations were conceived to read and, sometimes, write passive tags. **The operation initiative always belongs to the reader device.** If a two-way persistent connection is to be implemented on NDEF primitives, the reader will likely poll the tag often, until it gets some substantial payload. That's exactly the strategy of PHDC application protocol.

3.2 Android Development

Developing Android applications is mainly a platform-independent process. The programmers are free to use operating systems such as Windows, Linux, or Mac OS X. Accordingly, you may use any operating system you prefer. The software described in this book is not dependent on any OS and is usable on any one.

3.2.1 What Is Android?

Android is one of the world's most popular mobile platforms which is also used by mobile phones. Internally, it is a modified version of Linux with an integrated programming interface using the Java language. Android is owned by Open Handset Alliance, and it is indeed completely open sourced. Currently, Google leads the project, and most of the code is released under the Apache License. The application development environment includes a compiler, debugger, device emulator, and virtual machine named the Dalvik Virtual Machine (DVM) to execute the code.

If you are a Java programmer, you likely remember that Java codes are converted to byte codes and executed on a Java Virtual Machine (JVM) in the Java environment. Java byte codes, however, are not executed in Android. Instead, classes are complied into Dalvik executable files and run on the DVM. When you code your application, the Android SDK compiles the code into an Android package file with .apk extension which is to be uploaded and installed to the mobile device prior to execution.

Every Android mobile device has an installed Android OS version. The OS versions among different mobile devices may be different. When developing applications for specific Android devices, you should consider the platform version of that device. When you write an application for a targeted platform, some properties (for example, methods) may not be available for that specific version. Therefore, it is always a good practice to check the available resources in the application with the targeted platform's properties.

An Android application running on a mobile may perform all the services by itself or, alternatively, may use a previously available service on the mobile to perform some of the intended functionalities. For example, an application that requires a phone call may include the embedded code within the application or may fire — actually request firing — the already existing service on the mobile for this purpose.

As another example, if you implement an application that needs to record an image, it may make use of any available service that can take pictures. Consequently, you do not have to write any code for any service that is already available on the mobile. Your application simply activates the

camera component; the component takes the picture and returns the result to your application. Activation of the camera service and data exchange is seamless, so the user does not notice that the main application uses a separate camera service. In other words, it is assumed that the camera service is a part of your application.

The major components of the Android OS are the Linux kernel, Android run time, libraries, application framework, and applications, as shown in Figure 6.

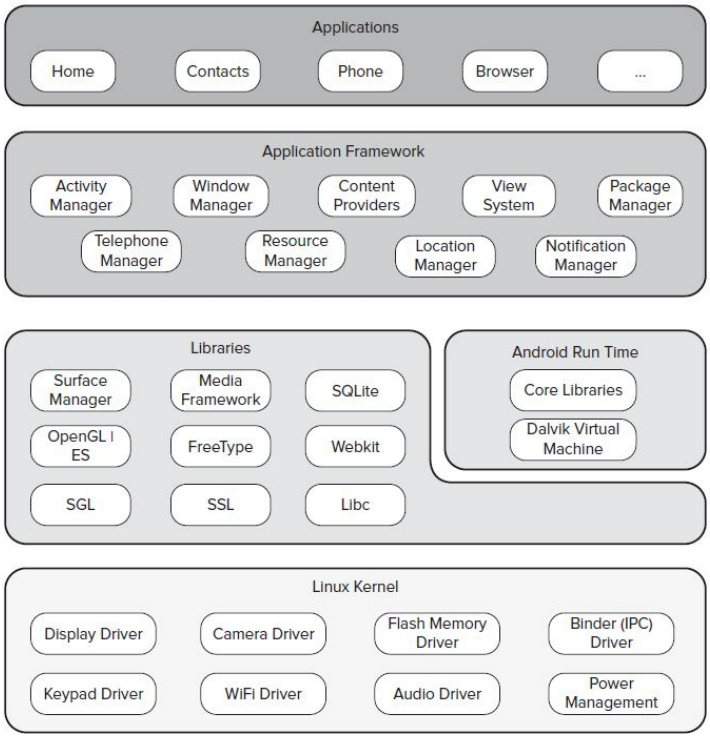


Figure 6: Core Libraries Of Android

3.2.2 Android Runtime

Android includes a set of core libraries that have 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 DVM. Dalvik enables a device to run multiple VMs efficiently. DVM executes files in the Dalvik executable (.dex) format, which optimizes memory use.

3.2.3 Libraries

Android includes a set of C/C++ libraries that provide the necessary capabilities for the Android system. These capabilities are provided to developers through the application framework. Some of the core libraries are shown in Figure 6.

3.2.4 Application Framework

Because Android provides an open development platform, it enables developers to build rich and innovative applications. Developers can benefit from various Android components and services, such as accessing device hardware, using location information, running background services, setting alarms, and adding notifications to the status bar.

All the developers have access to the framework API, and the application architecture is designed in such a way that an application can publish its capabilities and then others can make use of those capabilities.

3.2.5 Applications

Android includes a set of core applications such as an e-mail client, SMS program, calendar, maps, browser, and contacts. All these applications are written using the Java programming language.

3.2.6 Android SDK

For developers, the Android SDK provides a rich set of tools, including the debugger, libraries, handset emulator, documentation, sample code, and tutorials. Android applications can be easily developed using Eclipse (Android's official development platform). Eclipse provides rich features such as content assist, Java search, open resources, JUnit integration, and different views and perspectives for developing Android applications.

3.2.7 What You Need To Start

To develop Android applications, you need to install Java Development Kit (JDK) and Android Development Tools (ADT) Bundle.

Prior to installing the ADT Bundle to your computer, you should install JDK which also consists of the Java Runtime Environment (JRE).

4. SELECTED SOFTWARES

4.1 Eclipse Standard IDE 4.3.2 (Kepler)

Eclipse is an integrated development environment (IDE) that contains a base workspace and an extensible plug-in system for customizing the environment. Written mostly in Java, Eclipse can be used to develop applications. By means of various plug-ins, Eclipse may also be used to develop applications in other programming languages: Ada, ABAP, C, C++, COBOL, Fortran, Haskell, JavaScript, Lasso, Natural, Perl, PHP, Python, R, Ruby (including Ruby on Rails framework), Scala, Clojure, Groovy, Scheme, and Erlang. It can also be used to develop packages for the software Mathematica. Development environments include the Eclipse Java development tools (JDT) for Java and Scala, Eclipse CDT for C/C++ and Eclipse PDT for PHP, among others.

The initial codebase originated from IBM VisualAge. The Eclipse software development kit (SDK), which includes the Java development tools, is meant for Java developers. Users can extend its abilities by installing plug-ins written for the Eclipse Platform, such as development toolkits for other programming languages, and can write and contribute their own plug-in modules.

Eclipse is a platform that has been designed from the ground up for building integrated web and application development tooling. By design, the platform does not provide a great deal of end user functionality by itself. The value of the platform is what it encourages: rapid development of integrated features based on a **plug-in** model.

Eclipse provides a common user interface (UI) model for working with tools. It is designed to run on multiple operating systems while providing robust integration with each underlying OS. Plug-ins can program to the Eclipse portable APIs and run unchanged on any of the supported operating systems.

At the core of Eclipse is an architecture for dynamic discovery, loading, and running of plug-ins. The platform handles the logistics of finding and running the right code. The platform UI provides a standard user navigation model. Each plug-in can then focus on doing a small number of tasks well. What kinds of tasks? Defining, testing, animating, publishing, compiling, debugging, diagramming...the only limit is your imagination.

4.1.1 Open Architecture

The Eclipse platform defines an open architecture so that each plug-in development team can focus on their area of expertise. Let the repository experts build the back ends and the usability

experts build the end user tools. If the platform is designed well, significant new features and levels of integration can be added without impact to other tools.

The Eclipse platform uses the model of a common workbench to integrate the tools from the end user's point of view. Tools that you develop can plug into the workbench using well defined hooks called **extension points**.

The platform itself is built in layers of plug-ins, each one defining extensions to the extension points of lower-level plug-ins, and in turn defining their own extension points for further customization. This extension model allows plug-in developers to add a variety of functionality to the basic tooling platform. The artifacts for each tool, such as files and other data, are coordinated by a common platform resource model.

The platform gives the users a common way to work with the tools, and provides integrated management o f the resources they create with plug-ins.

Plug-in developers also gain from this architecture. The platform manages the complexity of different runtime environments, such as different operating systems or workgroup server environments. Plug-in developers can focus on their specific task instead of worrying about these integration issues.

4.1.2 Platform Structure

The Eclipse platform itself is structured as subsystems which are implemented in one or more plug-ins. The subsystems are built on top of a small runtime engine. The figure below depicts a simplified view.

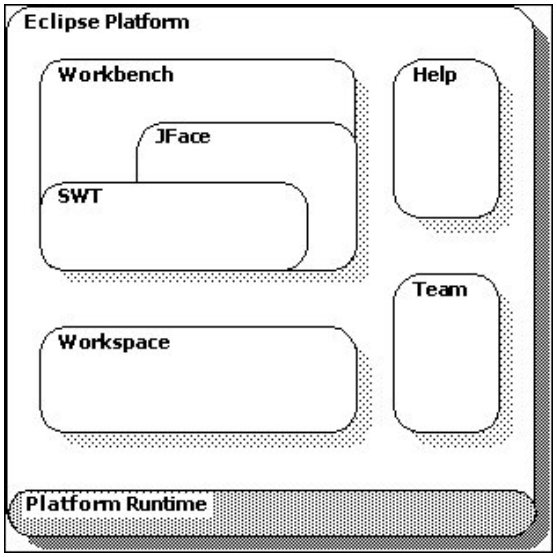


Figure 7: Structure Of Eclipse Platform

The plug-ins that make up a subsystem define extension points for adding behavior to the platform. The following table describes the major runtime components of the platform that are implemented as one or more plug-ins.

Platform runtime	Defines the extension point and plug-in model. It dynamically discovers plug-ins and maintains information about the plug-ins and their extension points in a platform registry. Plug-ins are started up when required according to user operation of the platform. The runtime is implemented using the OSGi framework.
Resource management (workspace)	Defines API for creating and managing resources (projects, files, and folders) that are produced by tools and kept in the file system.
Workbench UI	Implements the user cockpit for navigating the platform. It defines extension points for adding UI components such as views or menu actions. It supplies additional toolkits (JFace and SWT) for building user interfaces. The UI services are structured so that a subset of the UI plug-ins can be used to build rich client applications that are independent of the resource management and workspace model. IDE-centric plug-ins define additional functionality for navigating and manipulating resources.
Help system	Defines extension points for plug-ins to provide help or other documentation as browsable books.
Team support	Defines a team programming model for managing and versioning resources.
Debug support	Defines a language independent debug model and UI classes for building debuggers and launchers.
Other utilities	Other utility plug-ins supply functionality such as searching and comparing resources, performing builds using XML configuration files, and dynamically updating the platform from a server.

Table 1: Major Runtime Components Of Eclipse

4.2 Android SDK

The Android SDK contains the core tools needed to build and run Android apps. This includes the Android emulator, builder, docs and more. The Android SDK provides you the API libraries and developer tools necessary to build, test, and debug apps for Android.

Android SDK is a software development kit that enables developers to create applications for the Android platform. The Android SDK includes sample projects with source code, development tools, **an emulator**, and **required libraries** to build Android applications. Applications are written using the Java programming language and run on **Dalvik**, a custom virtual machine designed for embedded use which runs on top of a Linux kernel.

Every time Google releases a new version of Android, a corresponding SDK is also released. To be able to write programs with the latest features, developers must download and install each version's SDK for the particular phone.

The development platforms that are compatible with SDK include operating systems like Windows (XP or later), Linux (any recent Linux distribution) and Mac OS X (10.4.9 or later). The components of Android SDK can be downloaded separately. Third party add-ons are also available for download.

Although the SDK can be used to write Android programs in the command prompt, the most common method is by using an integrated development environment (IDE). The recommended IDE is Eclipse with the Android Development Tools (ADT) plug-in. However, other IDEs, such as NetBeans or IntelliJ, will also work. Most of these IDEs provide a graphical interface enabling developers to perform development tasks faster. Since Android applications are written in Java code, a user should have the Java Development Kit (JDK) installed.

The Android SDK includes the following:

- Required libraries
- Debugger
- An emulator
- Relevant documentation for the Android application program interfaces (APIs)
- Sample source code
- Tutorials for the Android OS

4.3 Android Development Tools (ADT)

ADT (Android Developer Tools) is a plugin for Eclipse that provides a suite of tools that are integrated with the Eclipse IDE. It offers you access to many features that help you develop Android applications quickly. ADT provides GUI access to many of the command line SDK tools as well as a UI design tool for rapid prototyping, designing, and building of your application's user interface.

Because ADT is a plugin for Eclipse, you get the functionality of a well-established IDE, along with Android-specific features that are bundled with ADT. The following describes important features of Eclipse and ADT:

- **Integrated Android project creation, building, packaging, installation, and debugging**

ADT integrates many development workflow tasks into Eclipse, making it easy for you to rapidly develop and test your Android applications.

- **SDK Tools integration**

Many of the SDK tools are integrated into Eclipse's menus, perspectives, or as a part of background processes ran by ADT.

- **Java programming language and XML editors**

The Java programming language editor contains common IDE features such as compile time syntax checking, auto-completion, and integrated documentation for the Android framework APIs. ADT also provides custom XML editors that let you edit Android-specific XML files in a form-based UI. A graphical layout editor lets you design user interfaces with a drag and drop interface.

- **Integrated documentation for Android framework APIs**

You can access documentation by hovering over classes, methods, or variables.

You can find the most up-to-date and more detailed information about changes and new features on the Recent Changes page at the Android Tools Project site.

5. SYSTEM DESIGN

5.1 MINIMAL REQUIREMENTS

The minimum requirements in terms of hardware, software, and programming skills to begin Android development are not very demanding.

Machine Requirements

Android development tools run on all flavors of computers of relatively recent vintage: Linux, Mac OS, or Microsoft Windows. An Android device such as a smartphone or tablet is useful (and of course the ultimate target for development), but is in fact not essential to getting started since the software contains virtual device emulators that allow you to develop and test.

Software Requirements

The software required for Android development is free and readily available on the Web:

1. The Java Development Kit (JDK).
2. The Eclipse IDE (Java Developers version) with the Android plugin is not technically essential since everything it does can be done using the command line, but it simplifies so many things that we will consider it to be essential.
3. The Android SDK and add-ons.

Programmer Requirements

Android is typically programmed using a combination of Java and XML (there is a Native Development Kit that permits programming in C/C++ for specialized tasks, but we will not address that here). Thus some background in these languages will be extremely useful. However, in the material that we will cover complete working code will be available for all examples. Therefore, anyone with some programming experience should be able to use the code as a starting point to learn how to program Java and XML within the Android environment.

5.2 SYSTEM REQUIREMENTS

Operating Systems

- Windows XP (32-bit), Vista (32- or 64-bit), or Windows 7 (32- or 64-bit)
- Mac OS X 10.5.8 or later (x86 only)
- Linux (tested on Ubuntu Linux, Lucid Lynx)
    - GNU C Library (glibc) 2.7 or later is required.
    - On Ubuntu Linux, version 8.04 or later is required.

Eclipse IDE

- Eclipse 3.7.2 (Indigo) or greater
- Eclipse JDT plugin (included in most Eclipse IDE packages)
- JDK 6 (JRE alone is not sufficient)
- Android Development Tools plugin (recommended)

6. UML DIAGRAMS

6.1 USE - CASE DIAGRAM

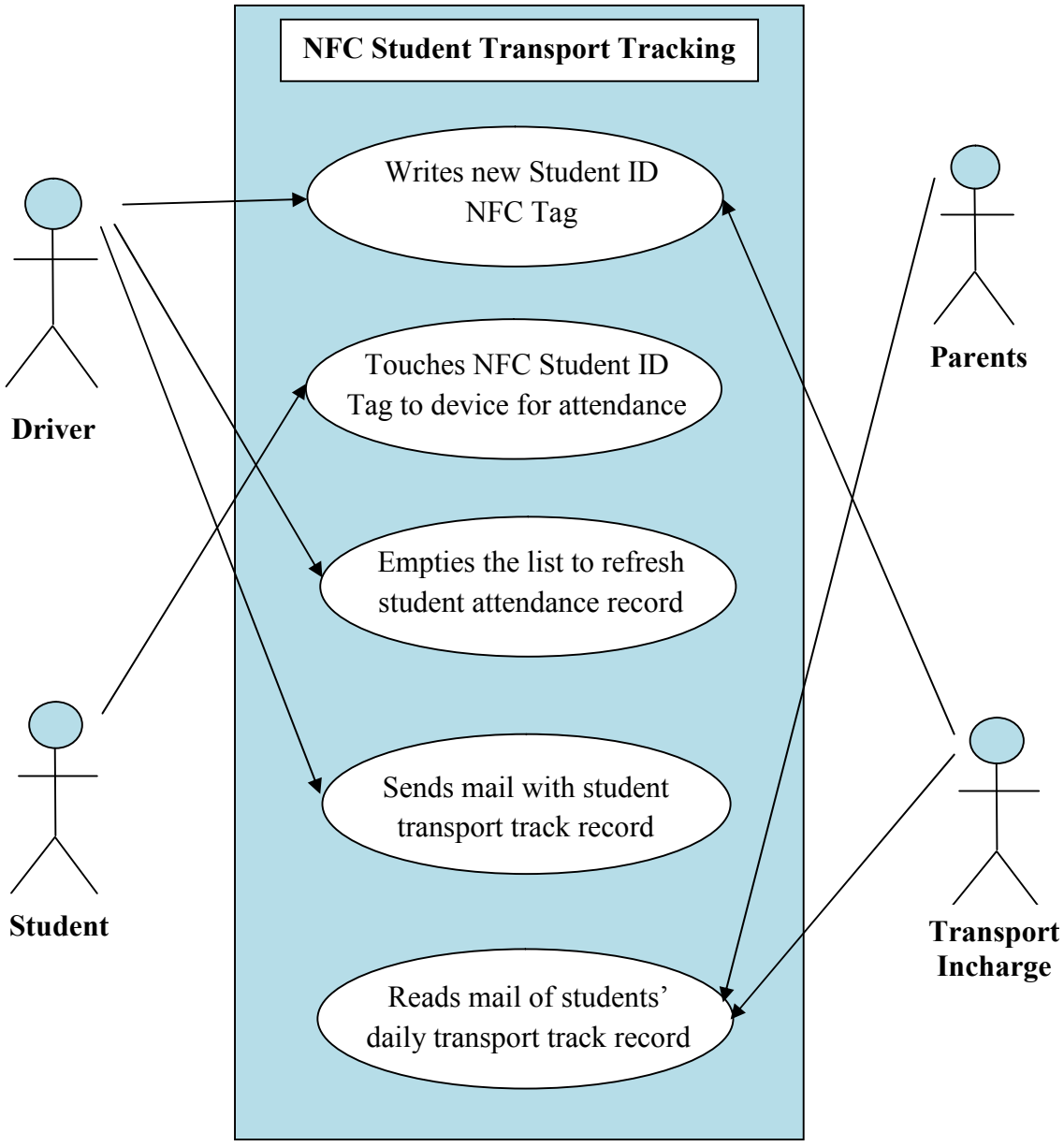
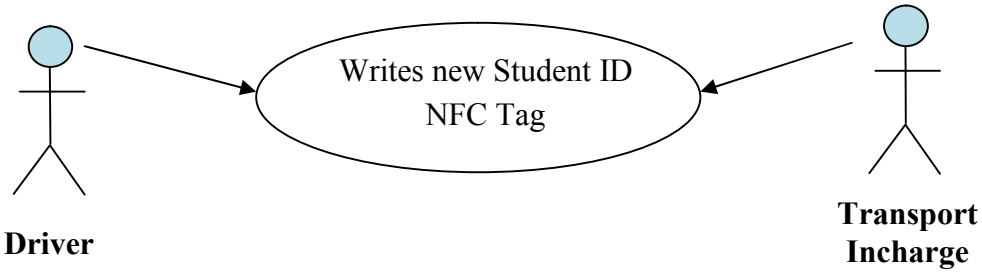
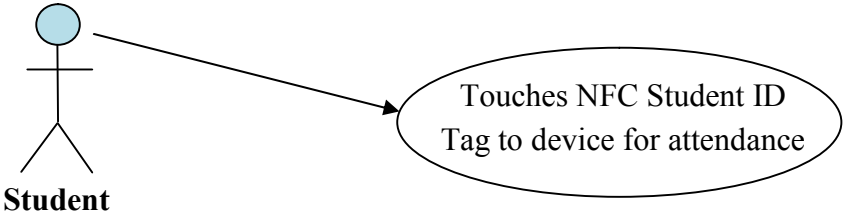


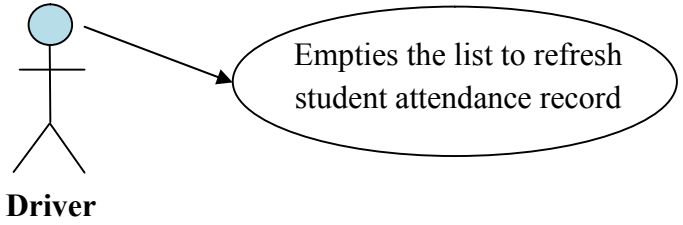
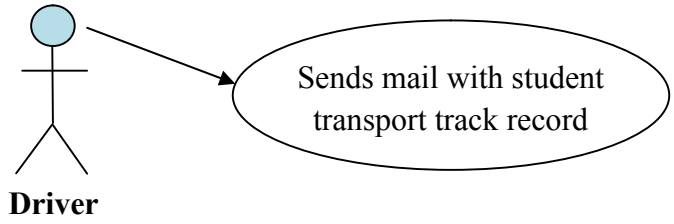
Figure 8: Use – Case Diagram For Student Transport Tracking

USE CASE DESCRIPTION

S.NO	USE CASE	USE CASE DESCRIPTION	ACTORS	BASIC FLOW	ALTERNATE FLOW	PRE CONDITION	POST CONDITION	RELATED USE CASES
1.	Writes new Student ID NFC Tag	Describes how a new transport user is registered.	Driver and Transport Incharge	A new transport user is registered by feeding his Enrollment No. and Name in a new NFC tag.	In case of NFC failure, the tag write operation will fail.	The new user must have a valid Enrollment No.	A new NFC ID tag is written for the new transport user.	-
2.	Touches NFC Tag to device for attendance	Allows the student to touch tag to device for his/her attendance.	Student	Student's attendance will be registered along with his/her getting-on time or getting-off time.	The attendance won't be registered if the student doesn't touch the tag properly to device.	The student should be a registered user and the tag should not be empty.	Registers the time of student getting on the bus when touched for first time, and getting-off time when touched second time.	-





3.	Empties the list to refresh student attendance record.	Allows the driver to empty and refresh the attendance list.	Driver	All records for the day's attendance are erased and the list is emptied.	-	There should be atleast one student record in the attendance list.	The student attendance record for the day is completely erased and refreshed.	-
								
4.	Sends mail with student transport track record	It allows the driver to send a mail through the application to the transport incharge for the day's transport tracking record.	Driver	Auto-generated mail sent to the transport incharge on just the press of a button.	Generates an error if no mail client is found.	Number should be already registered in the database.	A mail with the day's transport tracking record is sent to the transport incharge and the students' parents.	-
								
5.	Reads mail of students' daily transport track record	It allows the parents as well as transport incharge to read the day's transport tracking record.	Transport Incharge and Parents	An auto-generated mail is received by the parents and the transport incharge of the institution.	Mail sending operations may encounter problems if the correct mail client is not chosen from the	There should be atleast one record in the student tracking record and the sender's	The parents and the transport incharge can follow-up on the day's transport tracking and take	-

					'Activity Chooser'.	email address should be valid.	actions accordingly.	

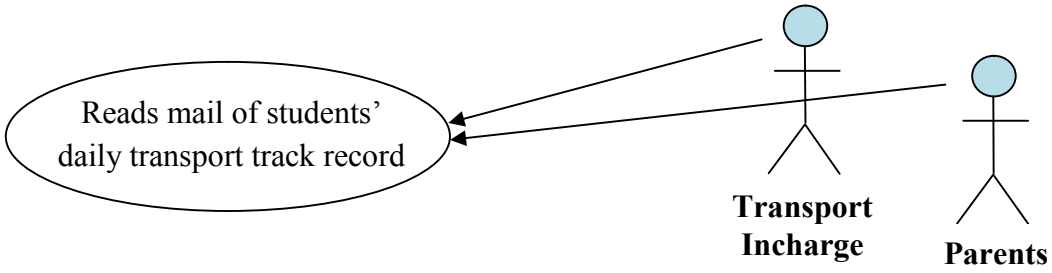


Table 2: Use Case Descriptions

6.2 WORKFLOW DIAGRAMS

6.2.1 TransportationActivity

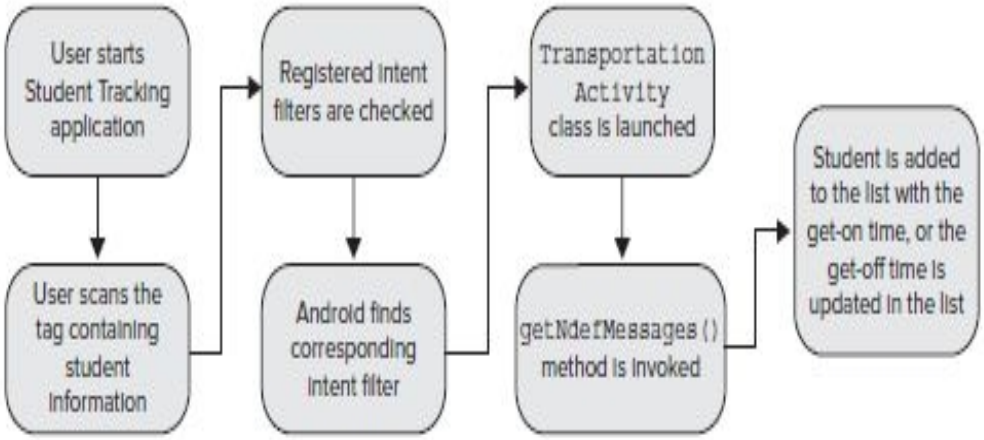


Figure 9: Workflow Diagram For TransportationActivity

6.2.2 TransportationWriterActivity

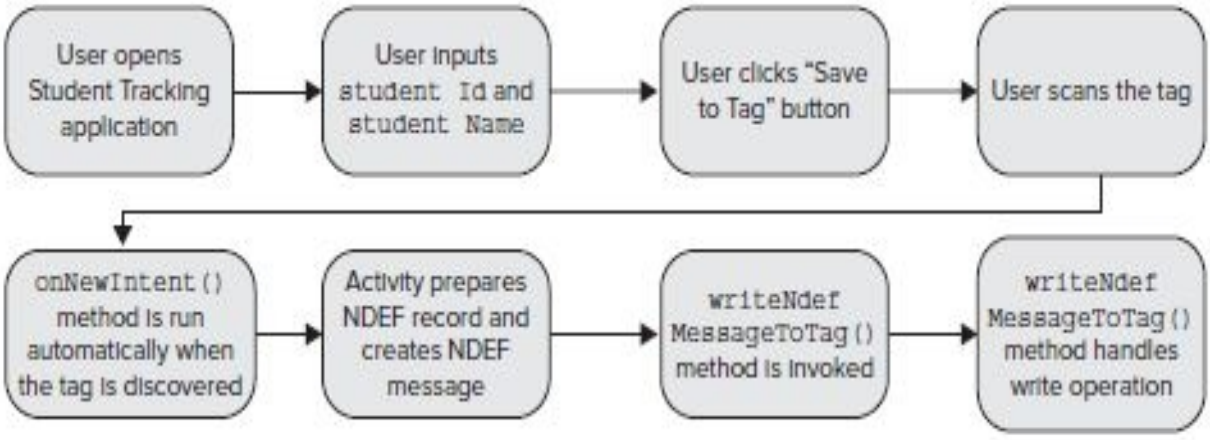


Figure 10: Workflow Diagram For TransportationWriterActivity

6.2.3 WebServiceActivity

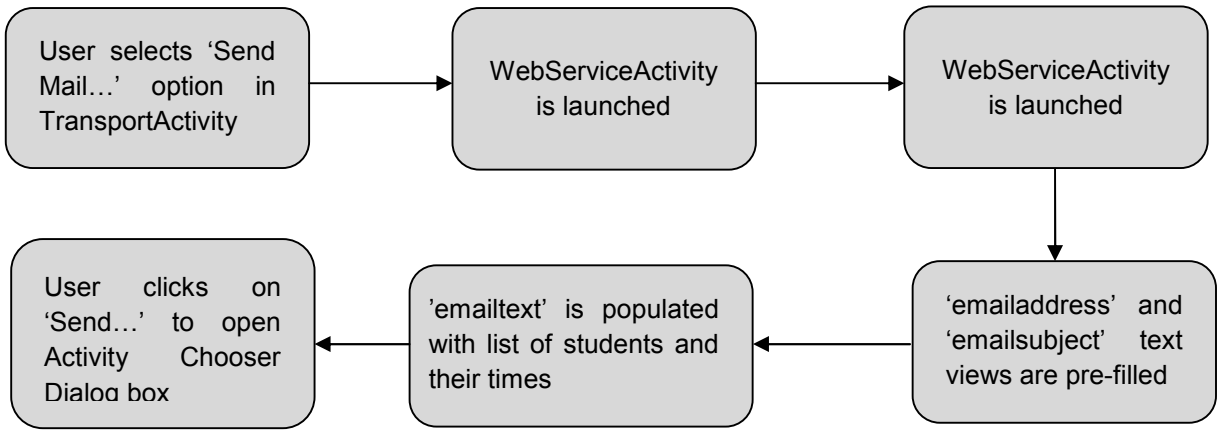


Figure 11: Workflow Diagram For WebServiceActivity

6.3 DATA FLOW DIAGRAM

6.3.1 Level-0 DFD

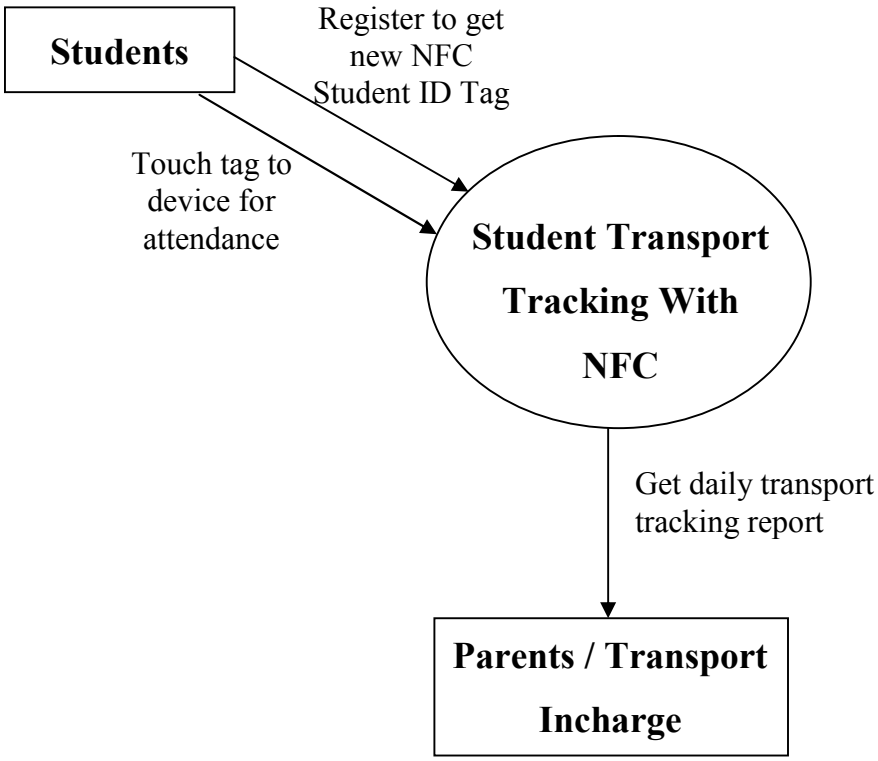


Figure 12: Level-0 DFD

6.3.2 Level-1 DFD

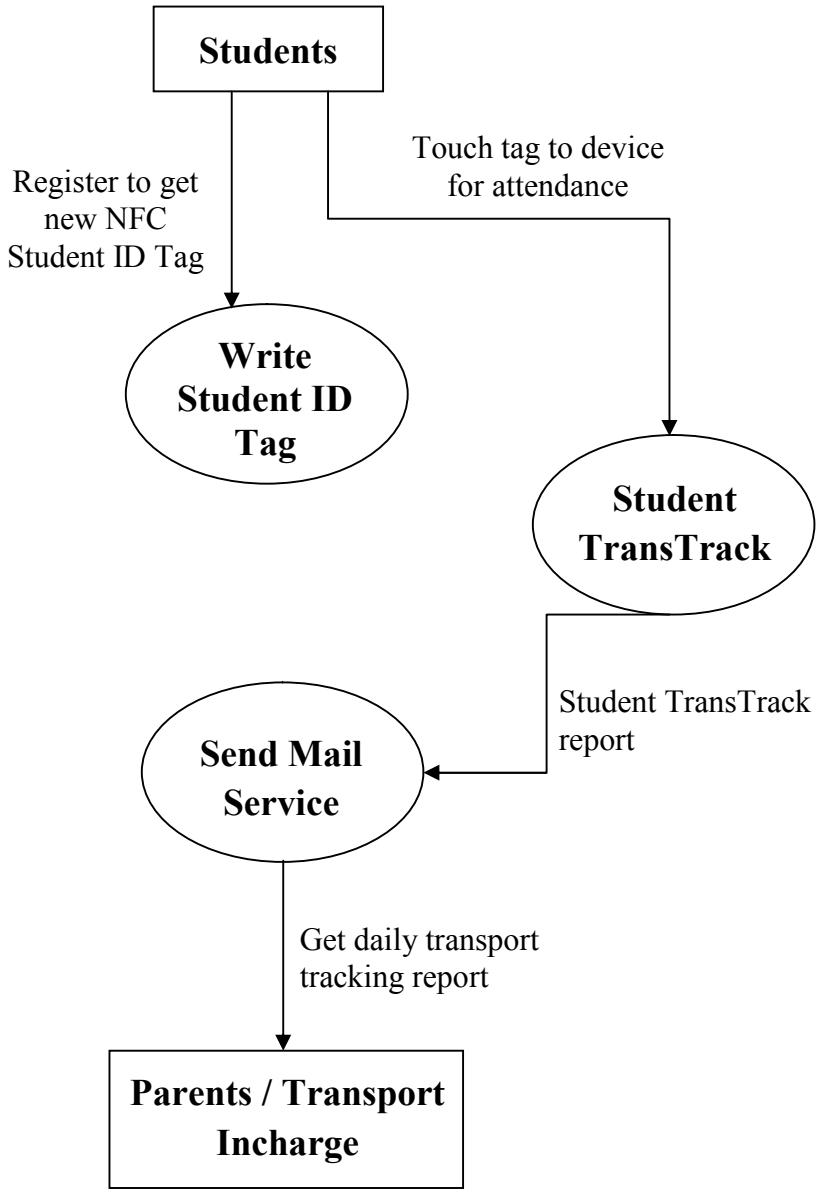


Figure 13: Level-1 DFD

7. TESTING

S. No.	Test Suite No.	Test ID No.	Description	Pre-condition	Input	Expected Output	Actual Output	Post-condition	Test Pass/ Fail	Remarks
1	1	1	**Write Tag Input**	If the right data is entered for Enroll No. and Name, it should be written to tag successfully.	Type Name and Enroll No.	'Message written to tag' toast.	'Message written to tag' toast.	Data written to tag after selecting 'Save To Tag'	Pass	–
2	2	1	**Student TransTrack**	On touching NFC tag to the device, student data along with timestamp should appear.	Touch NFC ID tag to device.	1 record appended to text view	NFC tag not read by device	Record not appended to text view	Fail	NFC should be switched ON in device.
3	2	2	**Student TransTrack**	On touching NFC tag to the device, student data along with timestamp should appear.	Touch NFC ID tag to device.	1 record appended to text view	NFC tag read by device	1 record appended to text view	Pass	–
4	2	3	**Empty List**	On clicking 'Empty List', the list should be cleared.	Click 'Empty List'	Empties the list and erases all records from screen	'Items deleted' message displayed	All records cleared from memory.	Pass	–
5	3	1	**Send Mail**	On clicking 'Send Mail' activity should show the	Click on 'Send Mail...'	Activity shows a mail draft with the	Drafted mail shown in	User clicks on 'Send...' to send mail.	Pass	–

				mail draft.		transport records of the day.	new window.			
7	3	2	**Send Mail**	After selecting 'Send...', activity chooser shows apps to launch mail service	Click on  Gmail	Redirects to 'Gmail' app to send auto-drafted mail.	Redirect operation failed	No auto-generation of mail.	Fail	Gmail app not installed on device.

Table 3: Test Cases

8. CONCLUSION

8.1 Some Final Thoughts

Near Field Communication is all about making our daily interactions easier – easier to pay for goods and services, easier to use public transport, easier ticketing for events, easier to interact with businesses, easier to share data between devices. At the heart of the technology is its ease of use, increasing convenience rather than enabling something new.

Most of the talk is focused on mobile payments and that has the potential to be a major driver of adoption. But as discussed above there are many other reasons for us users to want NFC too.

8.2 Future Scope And Improvements

The **'Student Transport Tracking With NFC'** project is all about making the daily hassle of students' transport tracking and attendance a lot quicker and tranquil. On one hand, it is so simple that even a driver who knows little about the dynamics of NFC can operate it. On the other hand, it is so effective that it has smooth transmission of data from one activity to the next, and can seamlessly record the days' transport activities and transfer it to the concerned authorities via mail in seconds.

However, there is always a scope of improvement and addition in any project. Each project must have a scope for the future so that the real advantage of developing the project can be taken. 'Student Transport Tracking With NFC' also has some scope for improvement:

- The project can be upgraded to be equipped with a database of past records of each student's transport track record.
- These records can also be accumulated to form a graphical view of each student's time records which can be studied to understand a pattern in student's daily timings (if any).

It is estimated that one in five smartphones will feature NFC technology by 2014, or almost 300 million NFC-enabled smartphones in three year's time. It will be a while longer until we're able to ditch our keys, wallet, cards and more in favor of a single device to rule it all. But beyond security concerns I wonder if I'll have to sleep on the porch in the event that my battery dies before coming home.

Jokes aside, NFC is certainly a technology to keep our eyes on as it has the potential to bring a new level of convenience to many aspects of a typical person's daily life. Are you looking forward to it?

9. APPENDIX

SCREENSHOTS

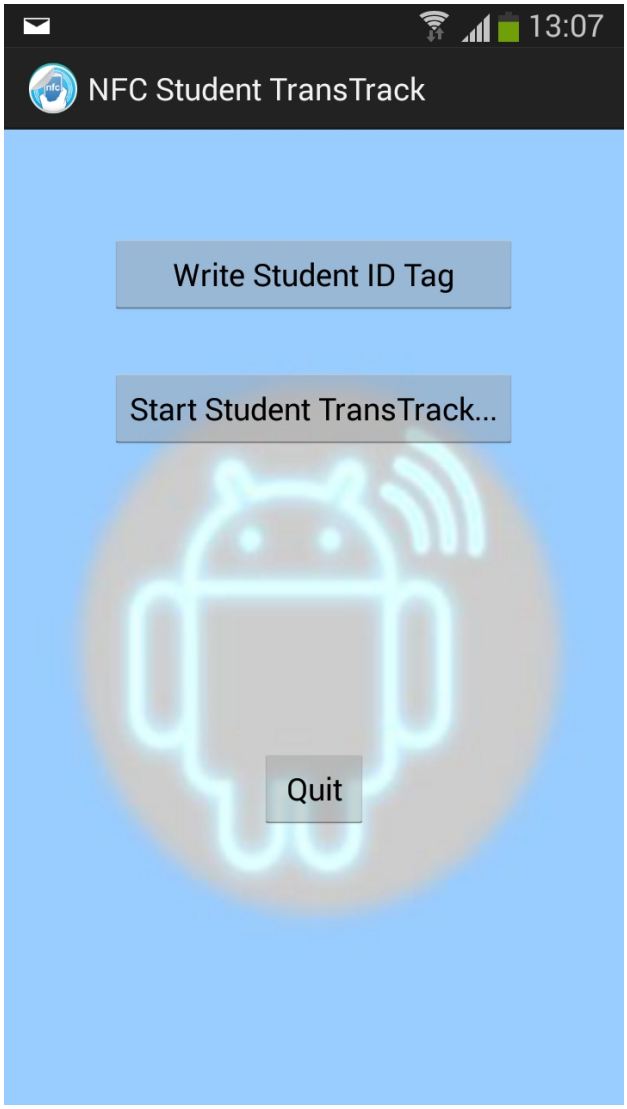


Figure 14: Home Screen Of 'NFC Student TransTrack'

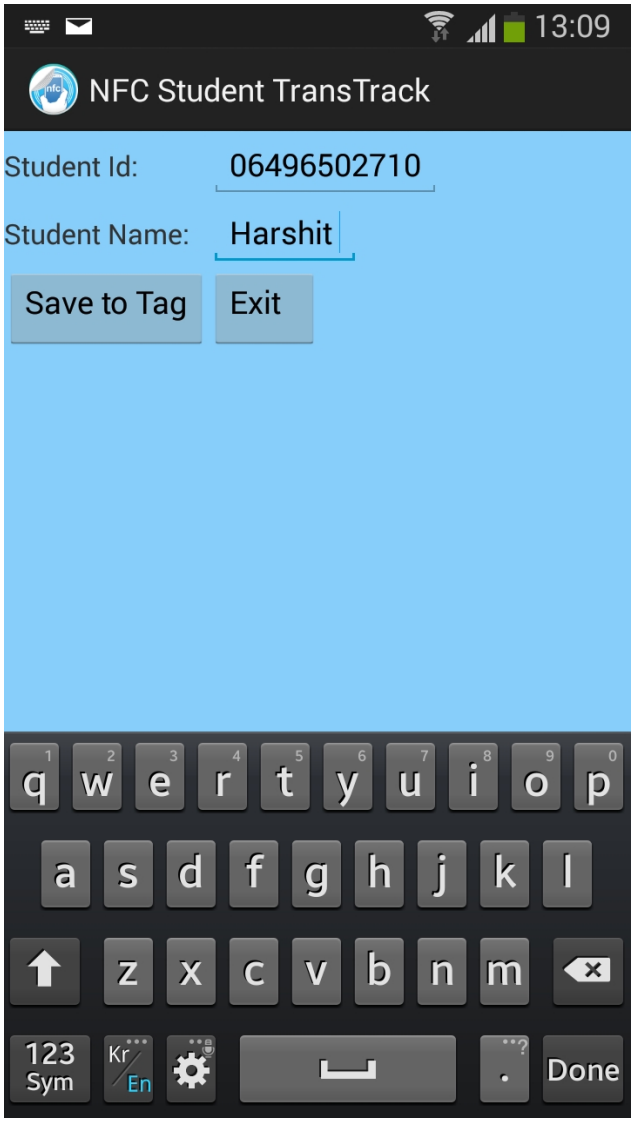


Figure 15: Launching of TransportWriterActivity on tapping 'Write Student ID Tag'

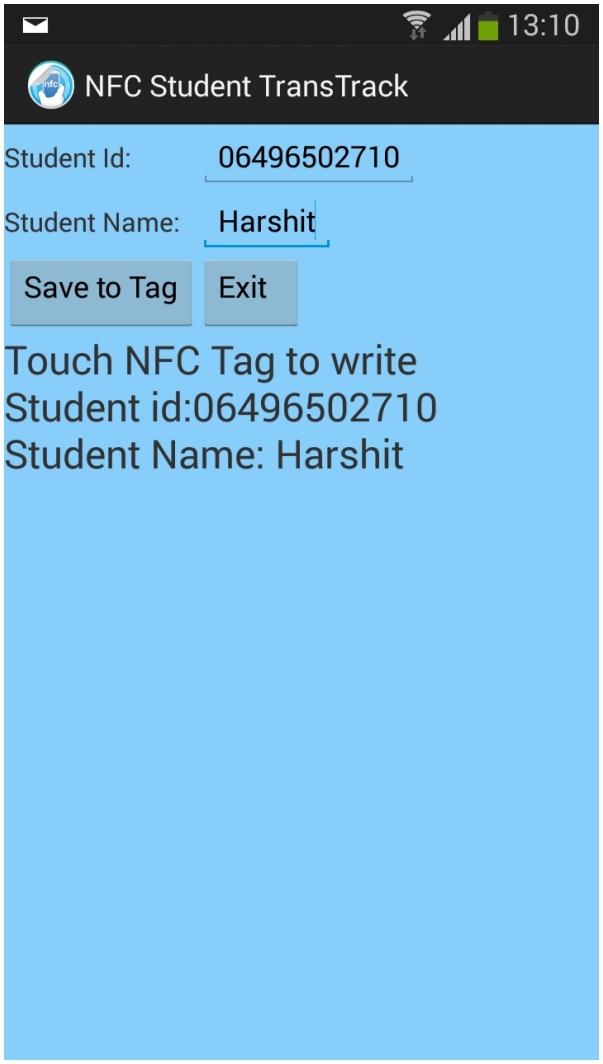


Figure 16: 'Touch NFC Tag to write' prompt on tapping 'Save To Tag'

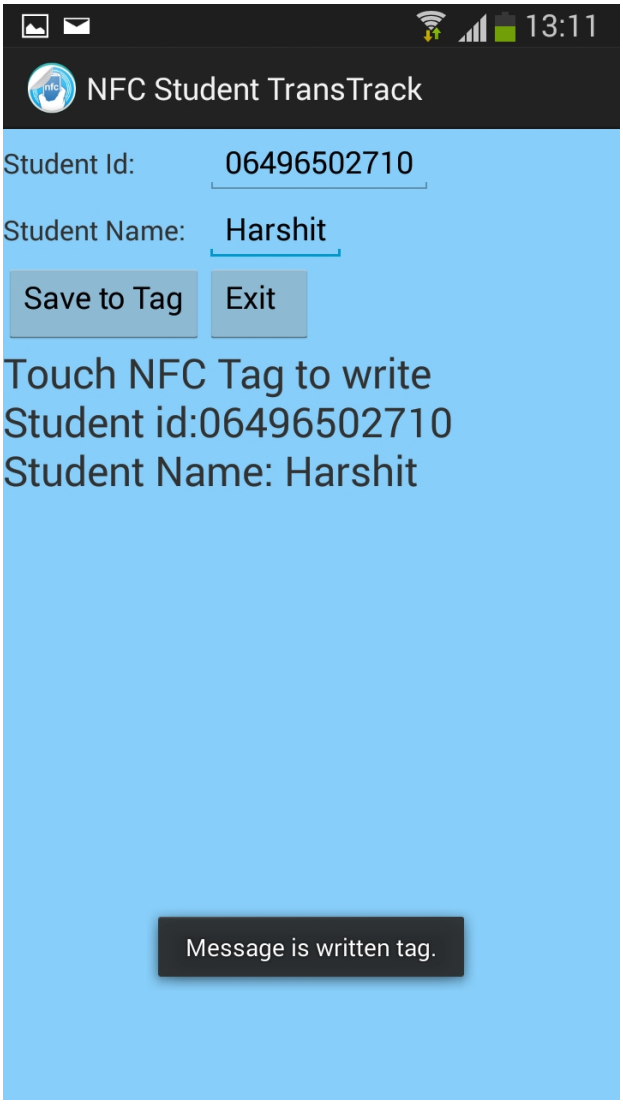


Figure 17: 'Message is written to tag' toast on succesful write operation

Figure 18: Launching of TransportationActivity on tapping 'Start Student TransTrack…'

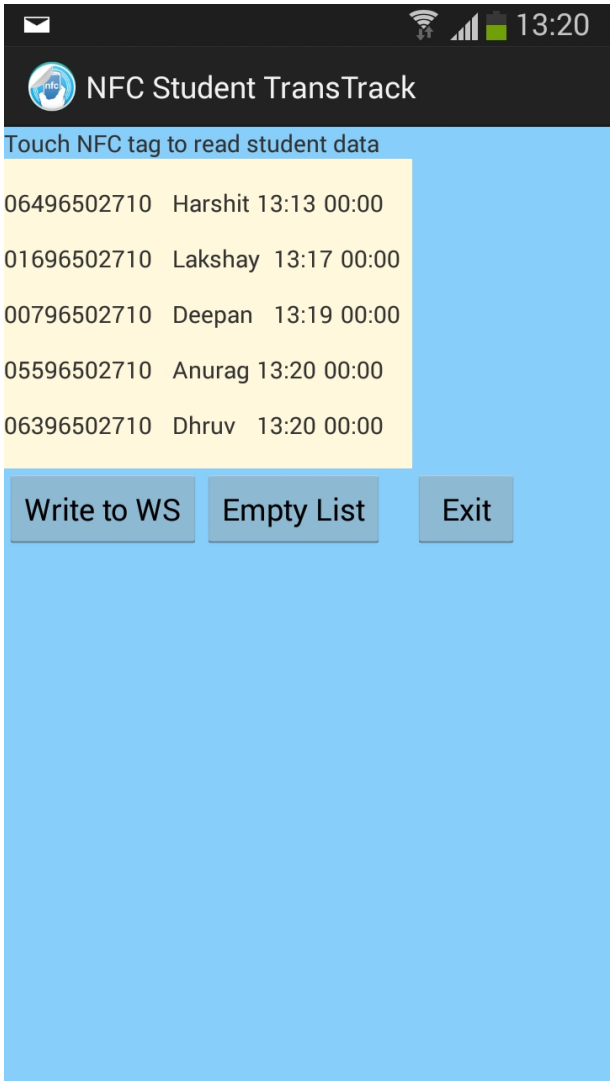


Figure 19: Adding of student getting-on information on reading NFC Tag for the 1st time

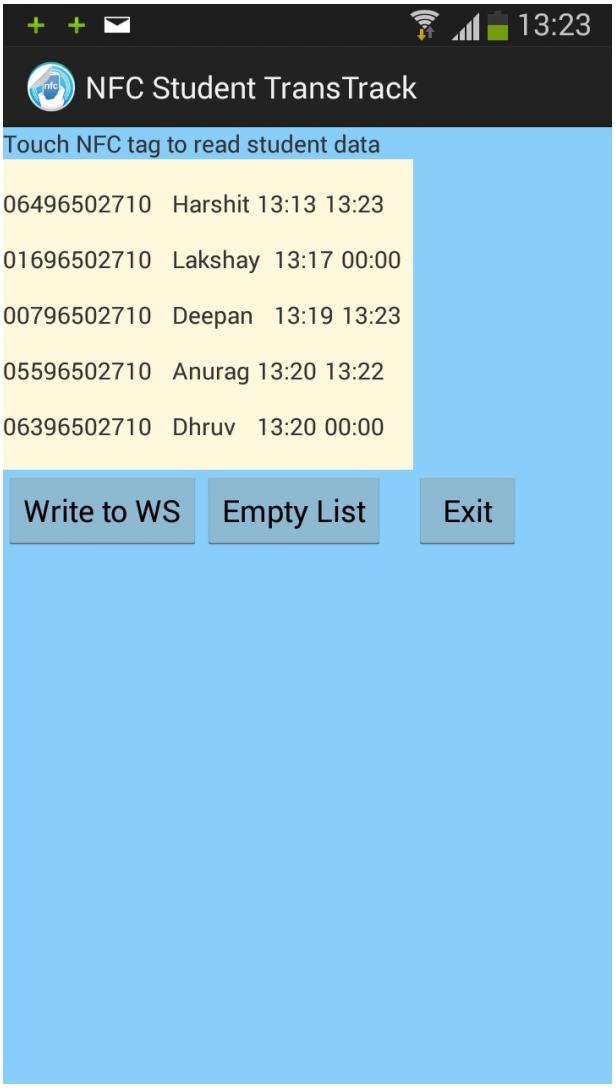


Figure 20: Adding of student getting-off information on reading NFC Tag for the 2nd time

Figure 21: Deleted all items from memory on tapping 'Empty List'

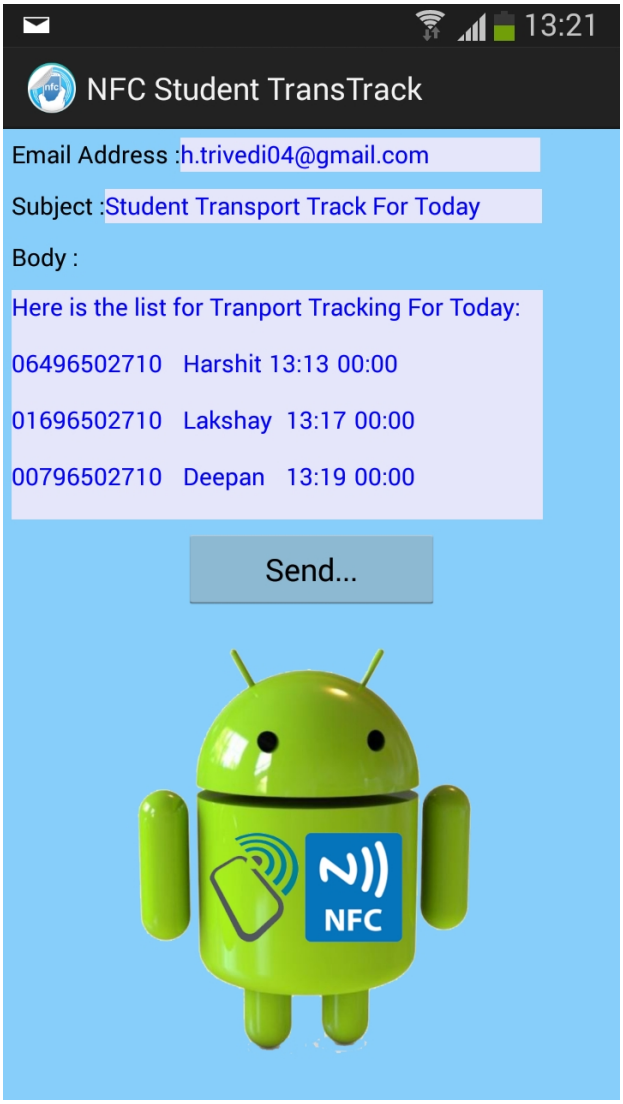


Figure 22: Launching of WebServiceActivity on tapping 'Send Mail…'

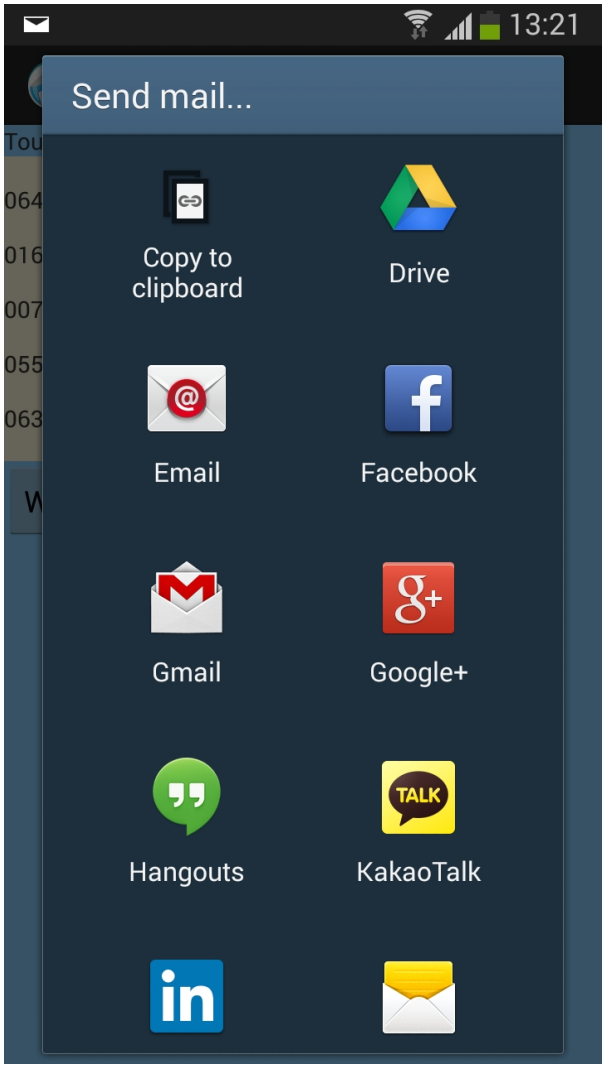


Figure 23: Activity Launcher to choose application for sending the e-mail

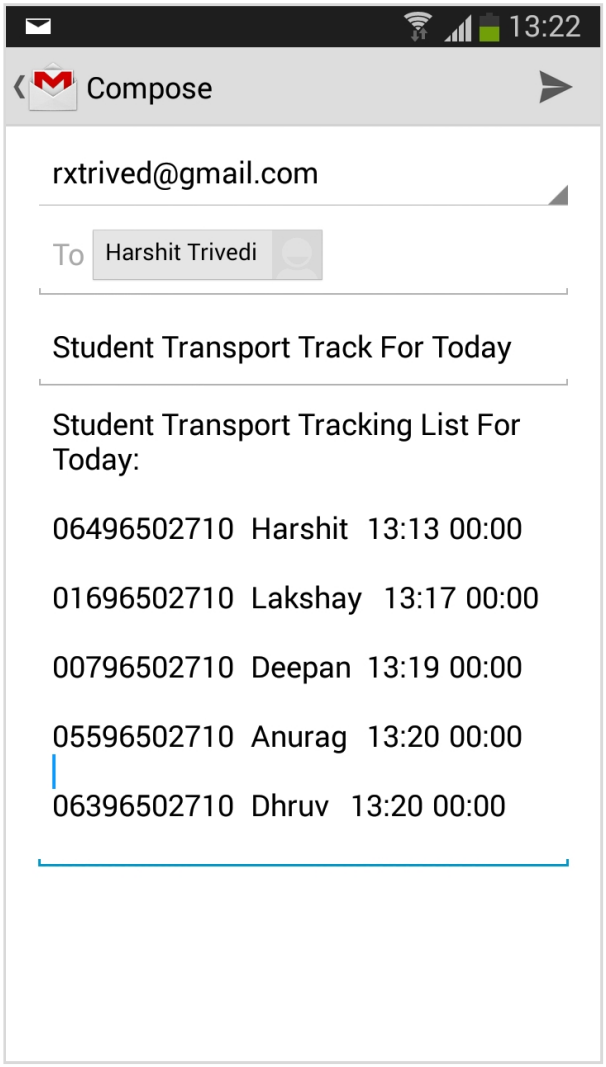


Figure 24: Auto-synthesized mail prepared on tapping 'Send…'

10. BIBLIOGRAPHY

[1] Jonathan Simon – Head First Android Development – O'Reilly Media, Inc. (2011)

[2] Vedat Coskun, Kerem Ok and Busra Ozdenizci – Professional NFC Application Development For Android – Wrox Programmer To Programmer (2013).

[3] Marc Knoll – TrendBlog.net – 18 Creative And Useful Ways To Use NFC Tags With Your Smartphone – http://trendblog.net/creative-and-useful-ways-to-use-nfc-tags-with-your-smartphone/ (February 8, 2014)

[4] Jose Vilches – TechSpot – Everything You Need To Know About NFC - http://www.techspot.com/guides/385-everything-about-nfc/page1.html (April 22, 2011)

[5] NearFieldCommunication.org© – http://www.nearfieldcommunication.org/ (2011)

[6] Tap Into NFC (in association with SeaDroid) – Blogspot.in – Tap Into NFC Workshop –

- Writing – http://tapintonfc.blogspot.in/2012/07/the-above-footage-from-our-nfc-workshop.html
- Reading – http://tapintonfc.blogspot.in/2012/07/nfc-workshop-series-how-to-read-nfc-tag.html

[7] Developing Mobile Applications That Support Near Field Communication (NFC) - http://wiki.developerforce.com/page/Developing_Mobile_Applications_That_Support_Near_Field_Communication_%28NFC%29