QReact

A Generic Framework to Create and Use QR Codes and a Usage Case in the Field of Access Control under Android

Author: *Atanas Plamenov Karaguiozov*,

Tutor: *Javier Fernández Muñoz*

"Most good programmers do programming not because they expect to get paid or get adulation by the public, but because it is fun to program"

Linus Torvalds

Special Thanks

"Everyone smiles in the same language"
George Carlin

It is difficult to list everyone who lent a hand in guiding me to this moment. These years have been full of adventures, twists, turns, mishaps... a journey of many hours and even more kilometres. Suffice it to say that this document has been worked upon in four different countries. But there were people who were always there to share these moments, here or a thousand miles away.

I want to give my heartfelt thanks:

- To all my family who were all there in their own ways and never let me go off the rails. No one can dream of being kept on their toes from a distance the way I was. A big dream of theirs is hopefully coming true today.
- To my tutors who were ready with suggestions and very clear ideas and did not let my imagination run far too wild.
- To Dr. Juan Enrique Muñoz Santiuste, without whom this project would have probably never come about. *¡Gracias!*
- They do say that university years count as the best in anyone's life. This second instalment of these would not have been the same without all the gang from the university. The time we spent together felt like it was never enough but you are part of what made everything the way it turned out to be. Thanks!
- Last and definitely not least, with all my love, to all my students from all corners of Spain and Portugal. There are way too many of you to list here and I will probably never find the right words to say how much I learned by teaching you. The beautiful moments we shared are every bit as appreciated as anything a teacher can receive without even deserving it. One more time, YOU GUYS ROCK!

Abstract

This project describes the development of a framework for secure exchange of secret information based on QR codes. The framework is programmed to be platform-independent. A possible usage scenario in the field of access control is described and a program to fit said scenario is presented, which runs on Android. Various design considerations are discussed and a number of possible off-the-label uses are considered. At the end, a road map for future improvements is presented.

The present document has been drawn up to show the steps in the development of the framework in detail.

Sumario

El presente proyecto describe el desarrollo de un framework para el intercambio seguro de información secreta basado en códigos QR. El framework se desarrolla independientemente de la plataforma operativa. Se describe un posible uso en el ámbito del control de acceso y se presenta un programa ejemplo de su uso bajo Android. Se sustenta el diseño elegido y se presentan algunos posibles usos en otros ámbitos. Al final, se presenta una posible vía de futura evolución de la plataforma.

El presente documento tiene como finalidad la presentación detallada de todos los pasos en el desarrollo del framework.

Contents

List of Figures

List of Tables

1. Introduction and objectives

"All programmers are genius playwrights
and all computers are lousy actors"
Programmers' folklore

This chapter briefly describes what motivated this project. The necessity for a system like the one developed is discussed and the various stages in its development are detailed. The chapter concludes with an explanation of how this document is structured.

1.1. What is *QReact*

QR codes have become ubiquitous in the recent years. Their ease of use, simplicity of scanning, speed of processing, robustness and, not least importantly, their information content are only some of the reasons why they have become a familiar sight for many. One aspect that makes them ideal for use in systems where certain access control is called for is the fact that they provide an accessible way of hiding information in plain sight. With an added layer of security to obfuscate their otherwise plain-text content, QR codes are an invaluable asset in similar applications.

Grosso modo, the objective of this project is to create a framework for access control based on QR codes. The framework, called *QReact*, defines *multi-layered QR codes* which can be used for *soliciting* and *transferring information.* The framework provides functionality related to the creation, validation, interpretation and possible execution of the instructions encoded in them. *Complete modularity* is a characteristic feature, allowing *QReact* to be *ported to other systems* and its *functionality expanded* effortlessly. Said design also does not pose a challenge to adding *extra functionality* when so required.

To branch out, *QReact* can be *used for a variety of other applications* with little or no modification.

In brief, these are the building blocks of this project:

- the framework. Various classes have been programmed which define the syntax of the instructions and of the configuration files. Two classes consisting almost entirely of static methods are programmed to aid the parsing process: these are used throughout the framework and allow seamless changes if these are needed.
- a GUI to aid in the creation of QR codes.
- a test application to illustrate the overall functionality in a specific usage scenario.

1.2. Why this framework

In the world of security and access control, it is desirable that secret information be transferred and interpreted securely. The volumes of information in question are not high, maybe a hash code is required or a password must be entered. The basic requirements are *speed* and *ease of use* and, if necessary, *ease in the modification.*

As an example, consider an electronic badge system, like the ones typically used in hospitals. Access is controlled by approximating the badge to the electronic reader. The badge itself, in many cases, is just a tamper-proof chip in which relatively static identification information is hard-coded. Said information can in principle be modified, depending on the technology used, but the process, while not difficult, is somewhat lengthy and definitely not immediate—special hardware is required for that. The fact that this example system is based on hardware has the advantage that it is almost entirely tamper-proof: if unauthorised access were to be gained, an intruder should know the internal workings of the chip and most probably have the necessary controller to properly re-program the chip in the badge. “Hacking” into such a system is certainly not impossible but it is labour intensive. A possible disadvantage is that the process of changing the identification information is cumbersome, time-consuming and potentially involves added expense: it consists of re-programming some or all of the chips or acquiring new ones. So while ease of use and speed are certainly a given with such a system, modification might not always be easy.

As another example, there are instances in which certain information is to be shared with only a certain group of people. For example, in an office there might be two WiFi networks: one for the general user and one internal, just for the use of the staff, possibly giving access to more online resources. As a safety precaution, the ESSID of the internal network is hidden and its pass code is changed, say, every fortnight. In this scenario, using hardware-based information exchange similar to the one described previously would be inefficient given the frequent changes to the information to be shared and the fact that every time that happens, the users would have to apply the changes manually.

These examples, and many more, show that addressing flexibility can often be a daunting task when it comes to sharing secret information. QR codes provide an easy, accessible way to tackle this problem. They also provide another layer of flexibility: *automation*. In the second example above, a little piece of simple software could read the QR code and automatically update the WiFi pass code as necessary.

Another way to use *QReact*, without modifying a single line of code in the framework, is custom-built one-time access systems, similar to the *GooglePlay* vouchers. This open truly immense possibilities, such as store coupons, one-time payment codes, electronic checks and many many more.

It is important, however, to realise that this framework does not accomplish security on its own. It provides a way to automate and simplify certain processes but still uses underlying security measures. In brief, *QReact's aim is not to replace existing access control systems but is presented as an alternative to them, providing easily implementable automation as an additional feature.*

1.3. Development process

There are six phases in the development of *QReact*.

Phase one consists of a study of existing alternatives and uses of QR codes. Attention is paid to their interpretation and generation. Finally, a brief analysis of platform requirements is presented and a platform is chosen to run *QReact* on.

Phase two is the analysis of system requirements. A number of quirks of the target operating system are considered.

Phase three deals with the detailed design of *QReact*. It is based on the system requirements laid out in phase two. Every part of the system implemented is analysed in order to make sure that all the requirements are met.

Phase four is the implementation of the actual system. Modularity is respected by testing every module individually.

Phase five consists of tests performed on the system as a whole.

Phase six is the development of this project report.

1.4. Project report: structure

1.4.1. Chapter one

Chapter one introduces the objectives and the need for this project. It also looks ahead and gives an overview of the development process and at what is to come in this project report.

1.4.2. Chapter two

Chapter two begins with an introduction to QR codes and why they have been chosen as an information vector. Next, a brief overview of the quirks of the operating system of choice is presented and how these will impact on the overall development of the framework.

1.4.3. Chapter three

An overview of the system requirements is presented. The project structure is considered in detail, explaining how the code design will fulfil the system requirements.

1.4.4. Chapter four

Chapter four deals with budgeting and time planning.

1.4.5. Chapter five

Chapter five presents an objective view of the possible future improvements to the system and some of its possible off-the-label uses. It also presents a subjective account of what was learned during the phases of analysis and development.

2. The State of the Art: Scannable Codes and Android

"Simplicity is prerequisite for reliability"
Edsger Dijkstra

Scannable or optical codes are all around us. From the simple bar codes we are so familiar with in supermarkets, bookshops, music shops and the like, to the stack codes we often find printed on our aeroplane tickets to the QR codes that have recently gained popularity with mobile phones, it is difficult to imagine an area of high-tech life where they could not be used.

This chapter describes some of the most widely-used optical codes, with special attention being paid to QR codes, of which various detailed aspects are considered. The chapter goes on to discuss why Android was chosen as the development platform and concludes with a review of some of the apps which use QR codes.

2.1. Introduction to scannable codes

At the most basic level, a *scannable code*, an *optical code* or a *bar-code*[1] is an optical machine-readable representation of data about the object to which it attaches. The

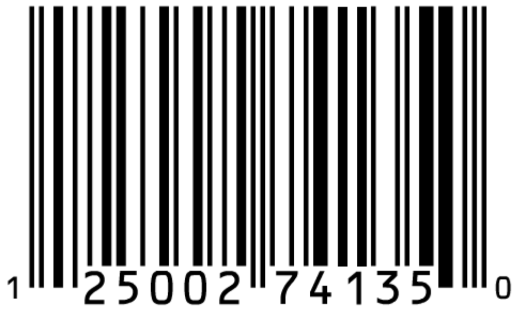
[1]As used in this context, a bar-code is a misnomer. Strictly speaking, bar-codes are just one of the many types of optical codes that exist.

encoding is done by combining patterns of different geometric complexity, the simplest being dots and parallel lines and growing in complexity up to intricate combination of these plus rectangles and hexagons. These codes are scanned by optical scanners and interpreted by special software. Optical codes take various *characters* as input and produce a *(code) symbol* as output.

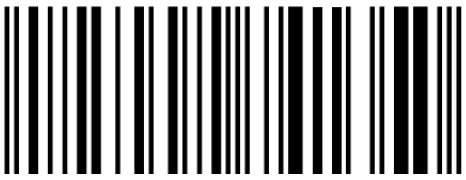
Linear codes

The optical codes with which we are most familiar are also the oldest in use: the bar-codes, also called *linear* or *1D codes*. They represent information using parallel lines of variable width. They were first used to label rail-road cars but their commercial success did not come by until they were used to automate supermarket checkout systems, which is where they are ubiquitous nowadays. The very first scanning of the now standard Universal Product Code (UPC) bar-code was on a pack of Wrigley Company chewing gum in June 1974[25]. The simplicity, universality and low cost of the bar codes have all limited the widespread use of other means of identification systems for different objects until the first decade of the 21st century, when the RFID systems were introduced.

Ordinary linear codes are vertically redundant. This means that the same information is repeated vertically, allowing for the heights of the bars to be truncated without any loss of information. This redundancy also allows a symbol with printing defects, such as spots or blank spaces, to be read and interpreted correctly. This is the protection linear codes provide against misreads. The higher the bar heights, the higher the probability that at least one path along the bar code will be readable. Figure 2.1 shows a bar code and its truncated version.



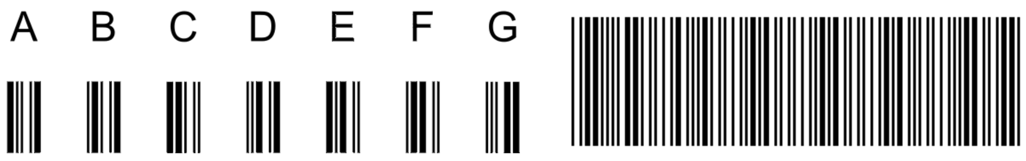
(a) Original bar-code with numerical values



(b) Truncated version of the same bar-code

Figure 2.1.: Illustration of the vertical redundancy of linear bar-codes. Both images have the same scale. The two bar-codes are completely equivalent

An ordinary linear code encodes numeric information only. As an example of their evolution we turn to *Code 39*. It was the first alphanumeric symbology to be developed and is still in use today, among others by United States Department of Defense, and the Health Industry Bar Code Council. It was developed at Intermec in 1974. Its original design included two wide bars and one dark to encode each character, resulting in 40 possible characters (see figure 2.2). Setting aside one of them, the asterisk, as a start and stop pattern left users with 39 characters, from where the code got its name.



(a) Character patterns (partial) (b) Example

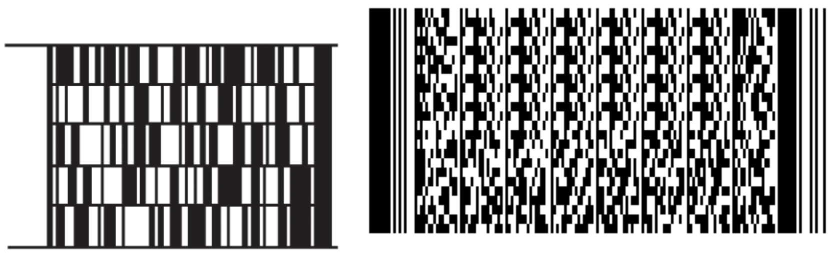
Figure 2.2.: Code 39

2D codes

Linear codes store information along the length of the symbol. 2D codes store information along the height as well as the length of the symbol. Thus both dimensions are used, which increases the amount of information that can be encoded. Using two dimensions also implies that the vertical redundancy is reduced. Therefore, to ensure protection against misreads, most two-dimensional codes use check words to ensure accurate reading.

There are three basic categories of 2D codes. *Stacked 2D codes* can be thought of as several linear codes stacked on top of one another. They therefore code the data in a series of bars and spaces of varying width. *Matrix codes*, on the other hand, code the data based on the position of black spots within a matrix. Each black element is the same dimension and it is the position of the element that codes the data. *Polar coordinate codes* use a similar idea, the difference being that in them the arrangement of the different parts of the symbol is circular. They have an additional element, a centre circle, known as 'the bullseye' which is used for calibration. From it, the different characters can be decoded by measuring angles from the bullseye.

There are well over 20 different 2D symbologies available today. A brief description of some of them follows.



(a) Code 49 (b) PDF 417

Figure 2.3.: Example of two stacked 2D codes

Code 49

The first truly two-dimensional code was introduced again by Intermec Corporation in 1988 when they announced *Code 49*. The idea was to pack a lot of information into a very small symbol. Each symbol can have between two and eight rows. Each row consists of a leading quiet zone[2]; a starting pattern; four data words encoding eight characters, with the last character a row check character; a stop pattern; and a trailing quiet zone. Every row encodes the data in exactly 18 bars and 17 spaces, and each row is separated by a one-module high separator bar (row separator). It had sufficient capacity to encode the complete ASCII 7-bit table.

PDF 417

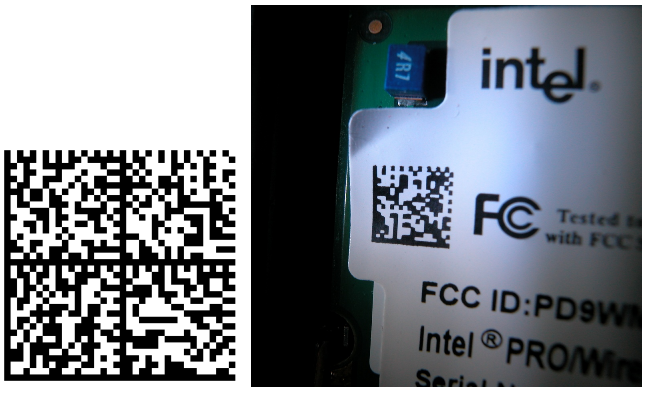
This is another very widely used 2D code, developed in 1991 by Ynjiun Wang at Symbol Technologies, now owned by Motorola. PDF stands for Portable Data File. Its symbology consists of 17 modules each containing 4 bars and spaces, hence the number 417. The structure of the code allows for between 1000 to 2000 characters per symbol with an information density of between 100 and 340 characters. Each symbol has a start and stop bar group that extends the height of the symbol. This code is now in the public domain. Applications include boarding passes for aeroplanes and other means of transport, security ID cards, inventory management. This format (together with *Data Matrix* outlined below) is used by United Postage Service (UPS).

[2]In the optical codes terminology, a quiet zone is a part of the code which is used only to indicate start or end of a section

Data Matrix

This 2D matrix code was originally developed by Siemens, is designed to pack a lot of information in a very small space. A Data Matrix symbol can store between one and 500 characters. The symbol is also scalable between a 1-mil square to a 14-inch square. That means that a Data Matrix symbol has a maximum theoretical density of 500 million characters to the inch! The practical density will, of course, be limited by the resolution of the printing and reading technology used. Typical Data Matrix symbol sizes vary from 8×8 to 144×144 cells, which can store up to 2,335 alphanumeric characters. Data Matrix uses different error-correction features, Reed-Solomon or convolutional codes, which help the message to be decoded even if the symbol is partially damaged.

The most popular application of this code is the marking of small items such as integrated circuits and printed circuit boards. These applications make use of the code's ability to encode approximately fifty characters of data in a symbol 2 or 3mm square and the fact that the code can be read with only a 20 percent contrast ratio. Also, it is becoming increasingly popular on labels (for example, to encode the serial number of computer hardware) and in the postal service, most notably DeutschePost, for digital postmark on letters. Curiously enough, although this code is a free standard, no free documents exist that explain the encoding process. These can be purchased from the ISO website [33]. Figure 2.4 shows an example of this code and its usage.



(a) Example (b) Usage

Figure 2.4.: Data Matrix code

ShotCode

This curious looking circular code was developed by High Energy Magic of Cambridge University in 1999 when researching a low cost method to track locations. The symbol is similar to a dartboard with a bullseye in the centre and data circles surrounding it. The decoding is achieved by measuring the angle and distance from the bullseye for each symbol. Because of the circular design it is also possible to detect the angle from which the code was read. An example, taken from ShotCode Wikipedia entry, can be found in figure 2.5.

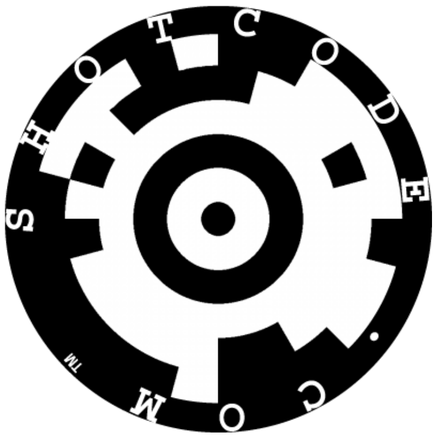


Figure 2.5.: ShotCode example

This code is designed to be readable by cameras found in mobile phones. Unlike other codes, ShotCode just contains URLs, not actual data. Two important features of this code are the speed with which the decoding takes place and the very small size of the program which does the job.

2.2. QR codes: a primer

A QR code (abbreviation for Quick Response code) is a matrix 2D code that consists of black modules arranged in a square pattern on a white background. It was Created by Toyota subsidiary Denso-Wave in 1994 and has only recently become one of the most popular types of optical codes.

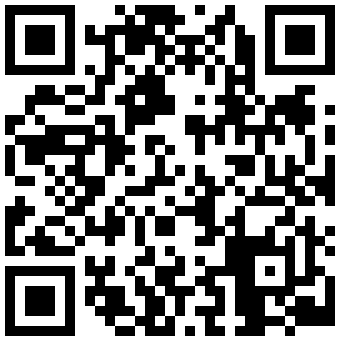


Figure 2.6.: An example of a QR code

2.2.1. Features and specifications

As any 2D code, it stores information in both horizontal and vertical direction which vastly increases its information storage capacity as compared to the linear bar-codes. It is suggested [1] that a typical QR code can encode the same amount of data as a bar-code in one-tenth of the size. For even a smaller printout size, the Micro QR code has been developed. In addition, reading and interpreting can happen at very high speed.

Figure 2.6 shows an example of a QR code. One can instantly recognize them by the three black squares placed at the corners of a square arrangement of dots. These are used for position detection. When a reader program scans the image, it tries to find these three squares plus an additional smaller, less visible square with a dot in the middle used for alignment (visible towards the bottom right-hand corner in figure 2.6). Only then can the contents be interpreted. These squares are always placed in the same positions within the matrix, which allows QR codes to be scanned at any angle, a similarity they bear to the polar coordinate codes described in the previous section.

The outer zone of the code is used as a quiet zone (see previous section for a definition). The version information is contained in a rectangular pattern on top of the bottom right and directly to the left of the top right square. Information about the format is contained around the top left square, directly to the right of the bottom left square and directly below the top right square. The line pattern that connects the three squares is required and can vary depending on the rest of the information. It is used for timing while scanning. The rest of the symbol contains the actual encoded information.

Numeric only	7089 characters
Alphanumeric	4296 characters
8-bit binary	2953 bytes
Kanji, full-width Kana	1817 characters

Table 2.1.: Maximum capacity of a QR code

Capacity

One of the most important features of this code is its capacity to encode various types of data. QR Code is capable of handling numeric and alphabetic characters, Kanji, Kana and Hiragana characters (let's not forget it was developed in Japan), symbols, binary, and control codes. A maximum of 7089 characters can be encoded in one symbol. Table 2.1 shows maximum data capacity for the different formats.

It should be noted, however, that not all scanning applications support the maximum amount of data.

QR codes have different versions which define the different sizes used. They start with version 1, which has 21×21 elements and goes on to version 40 sized at 177×177 elements. Each version adds four more elements to the previous, both horizontally and vertically. So version 2 would have 25×25 elements, version 3 would have 29×29, etc. Each symbol contains the full capacity according to the amount of data for the given format. As the amount of data increases, instead of cramming more dots inside the same symbol, thus lowering their clarity and potentially risking misreads because of resolution artefacts, more modules can be added to accommodate the data. These new modules are QR codes themselves and are a very ordered way to create new and larger symbols. This curious property is known as the *Structured Append Feature* of the QR codes and is illustrated in figure 2.7.

These modular symbols are created sequentially, i.e. when a new module is required, the information in the rest of them is repositioned so as to use all the capacity of the new symbol and positioned in a way that the information encoded in the second mini-symbol follows the one encoded in the first, the third follows the second, etc. Scanning these modular symbols is done considering the whole symbol as one, just as if it were a normal code symbol. The presence of the additional square markers in the individual modules helps the alignment process, as there are more points to scan and the probability of a misread would increase without bound without them. These additional squares also help the decoding process: they serve as separators between the different chunks of information

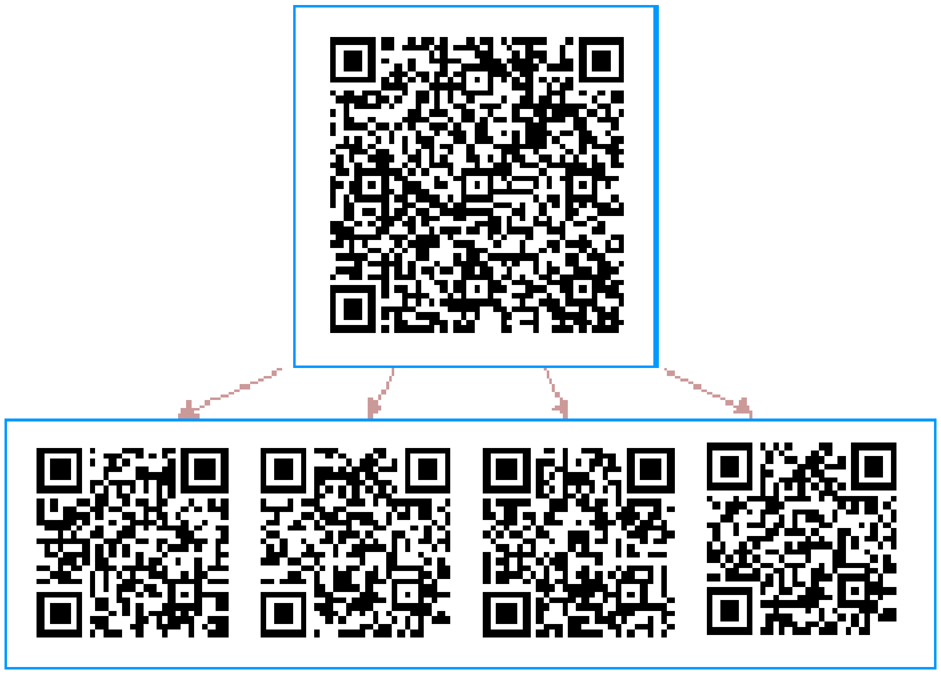


Figure 2.7.: An example of the *Structured Append Feature*

in the original message. Once these chunks have been decoded, the original message is composed by concatenating them starting from the top left and proceeding horizontally.

This interesting feature means that information stored in multiple QR Code symbols can be reconstructed as single data symbol. Furthermore, one data symbol can be divided into up to 16 symbols, allowing printing in a narrow area.

For further detail on the sizes and versions, one should consult [11].

Error correction capabilities

As mentioned in section 2.1, all scannable codes employ some kind of error correction mechanism. For linear codes, it is the vertical redundancy illustrated in figure 2.1. For 2D codes, error correction is introduced as controlled redundant data interwoven with the text itself, suggesting the use of digital techniques to accomplish this important feature. In the case of QR codes, it is accomplished using Reed-Solomon codes.

Reed–Solomon (R) codes were first introduced in 1960 by Irving Reed and Gustav Solomon, then members of the MIT Lincoln Laboratory. They are a non-binary variation of the widely used cyclic error-correcting codes. R codes can detect and correct

Level L	7%
Level Ms	15%
Level Q	25%
Level H	30%

Table 2.2.: Error correction levels of QR codes (percentages are approximate)

multiple random errors. By adding t check symbols to the data, R codes can detect any combination of up to t erroneous symbols, and correct up to $\lfloor \frac{t}{2} \rfloor$ of them. R codes add redundancy of twice the number of codewords that are to be corrected. They are also suitable for the correction of multiple-burst bit-errors.

Mathematically speaking, in R codes source symbols are viewed as coefficients of a polynomial $p(x)$ over a finite field. Today, encoding symbols are derived from the coefficients of a polynomial constructed by multiplying $p(x)$ with a cyclic generator polynomial[3]. This gives the idea of an efficient decoding algorithm, which was discovered by Selwyn Sampler and James Massey, and widely used today.

R codes have a great variety of uses in the modern digital world, although slowly being phased out by the more modern turbo codes and low-density parity-check codes. In data storage, R codes gained a lot of recognition in 1982 with the mass production of the compact disk, where two different R codes were interwoven. Similar schemes are used in the DVD and DAT products. The distributed online file system *Kuala* also uses them when it splits files into chunks. In data transmission, R codes were used, concatenated with convolution codes, to encode the digital pictures sent by the *Voyager* space probe. Since then, they have been used in a number of space missions, including the *Mars Pathfinder, Galileo, Mars Exploration Rover* and *Cassino's*. Another important use is in slid systems.

The error-correction capability of QR codes makes extensive uses of Reed-Solomon codes. This allows data to be restored even if the symbol is partially destroyed or dirty. Users can choose from four available error correction levels depending on the operating environment. Each level consecutively improves the error correction capability but also increases the amount of data and consequently the symbol size. Table 2.2shows the four available levels.

Levels Q or H may be selected, for example, for factory environment where the likelihood of a code getting dirty increases. For a clean environment, level L may be selected if we are after a large amount of data. Level Ms is the most frequently selected one as it provides a

[3]In this sense, they are reminiscent of BKcodes

trade-off between redundant data and protection against errors. As a somewhat simplistic example of how to determine which level to use, let's consider that we wish to encode 200 text characters and want to be able to correct 100 of them. The R code will add a redundancy of 200 characters (see above), so the total number of characters to encode becomes 400. Since we want to correct 100 of them, the fraction becomes $\frac{1}{4}$ or 25%. This suggests we need to use level Q protection to achieve our goal. If we wished to correct just 50 characters, the size would become 300 in total and the fraction would be $\frac{1}{6}$, which is close to 15% and so we would choose level Ms.

Micro QR code

For applications that require small amount of data, smaller space and that do not have the ability to handle larger scans, *Micro QR* codes were created. Data encoding in them is more efficient given that they only have one position detection pattern (see figure 2.8).

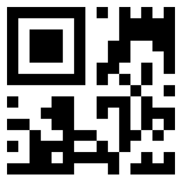


Figure 2.8.: An example of a Micro QR code

A typical symbol is a lot smaller than even the smallest version of a regular QR code. However, the capacity is also drastically reduced–for a Micro QR code, the maximum capacity is just 35 alphanumeric symbols. Error correction is limited to L, Ms and Q levels.

Copyright note

QR codes are property of *Denso Wave* and even the word "QR Code" is registered trademark in Japan, USA, Australia and Europe. Despite that, *Denso Wave* choose not to exercise their right to receive royalty for their use. The license to the use of the QR Code stipulated by HIS (Japanese Industrial Standards) and the ISO are not necessary. The specification for QR Code has been made available for use by any person or organization. *Dens Wave* do stipulate, however, that in order to use the word *QR Code* in publications, web sites, etc. an indication should be made that *QR Code is registered trademark of DENSO WAVE INCORPORATED*. This only applies for the word *QR Code*, and not for the actual image. This actually means that everyone can use QR codes freely and is no doubt one of the reasons why their popularity has soared recently.

Calculating the area of the QR symbol

The area of a printed QR symbol can be crucial to the application. Below is an illustrative example of how to calculate this important parameter and what to do is it is too large. Note that these calculations include the margin (the quiet zone, which as we know is a part of the symbol).

If we wished to encode 100 alphanumeric characters, these are the steps to follow.

1. Specify the error correction level and decide on a version by finding the intersection of alphanumeric characters and the chosen error correction level from the table in [11]. In our case, suppose we wish a standard error correction level Ms. The appropriate version is Version 5, as Version 4 with Level Ms holds only 90 characters.
2. If we use a printer with a 400 dpi resolution (pretty standard) and print with a 4 dot configuration, we use the equation

   $$\frac{25.4\text{mm/inch}}{400\text{dpi}} \times 4\text{dots/module} = 0.254\text{mm/module}$$

   There are 37 modules in Version 5, therefore the (horizontal) size of QR Code will be

   $$37\text{modules} \times 0.254\text{mm/module} = 9.398\text{mm}$$

3. Next, we secure a four-module wide margin on both sides, a requirement of these codes. For the purpose of these calculations we will use 8 modules, four on the left and four on the right. Now

   $$9.398\text{mm} + 0.254\text{mm/module} \times 8\text{modules} = 11.43\text{mm}$$

In other words, the required QR Code area is 11.43mm^2

If the QR Code area obtained in the process above does not fit the printing space, a decrease in the symbol version may be considered. Another idea is to make the module size smaller or to split the symbol.

2.2.2. Possible uses and impact

Given that QR codes can encode just about anything, their potential is limitless. In this section we discuss some of their potential uses and the impact they might have on them.

As QR codes were created in Japan, they quickly became very common there and are still in wide use. Other places where they have seen frequent use include the Netherlands and South Korea. Even though the rest of the world has been somewhat slower in their adoption, nowadays they are becoming more and more common. [2]

Examples of use in management settings

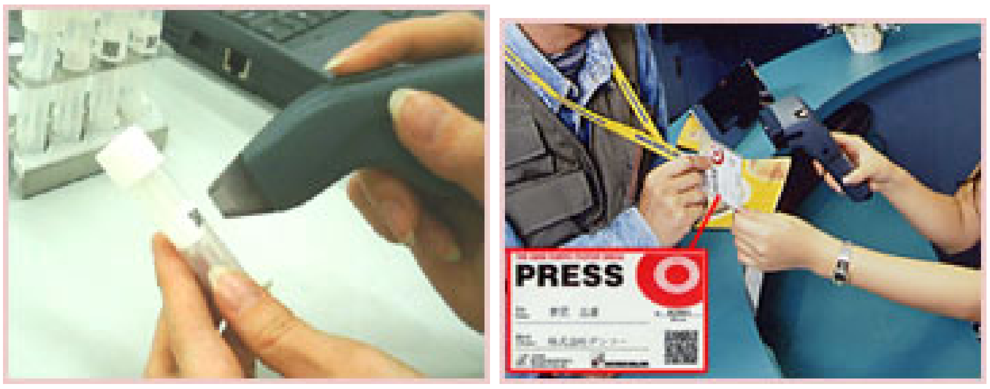
An example application of a Micro QR code could be tagging printed circuit boards or electronic parts. Micro QR Codes are small enough and can encode enough information for a serial number, which is typically up to 20 alphanumeric characters. At just 3mm^2, a manufacturer has plenty of room to attach it. The data can be used for process control, history control and automatic set-up.

Another possible use is in bookkeeping, particularly in larger libraries. Micro QR Codes can be used to identify ISBN used for books and ISSN used for periodicals, articles, etc. These entries require 13 and 8 digits, respectively. All that is needed is a little room on the spine of the book, as opposed to the front page, where to print the Micro Qr code. Then it would be possible to scan the periodical or book directly from the shelf where it lives.

For shipping slips and receipts in the automotive industry, a QR Code might contain customer data, shipper data, product number, quantity, etc. The data is used for ordering and product scanning. This system offers the benefit of gathering large volumes of shipping data by one-touch operation. Additionally, the decoded data is already in text form, which significantly reduces the cost of forms compared with conventional slips filled in using OCR software.

For logistics, product code, expiration date, manufacturing history, and other data can be encoded into QR Code. This enables first-in first-out execution based on expiration date control and improved traceability based on manufacturing history control.

Shipping companies can also benefit from the use of QR codes, in which shipping destination, product code, colour, size, and other data can be encoded and then printed on shipping instructions. The data is used for shipping control thus potentially preventing



(a) Test specimen management system (b) Access control system

Figure 2.9.: Two further uses of QR codes

shipping mistakes. It also enables instant gathering of shipping instruction data using handy terminals.

Sales can also benefit from QR Codes. Lots of additional information can be provided, such as warranty, care and handling of the product, service points, etc., which might be used for sales management purposes. This is particularly effective if using small-sized codes on small items given that a printed instruction sheet would be difficult to fit in the wrapping. This also enables efficient analysis of sales situation.

Other uses include test specimen management systems (for example in a microbiology laboratory) and access control systems as in figure 2.9. QR codes can also be used for inventory, where the symbol might contain information about property data, model numbers, user names, and usage locations.

Direct marketing and tracking

An almost immediate application of QR codes, following their release from the confines of Toyota, was in direct marketing. The idea was to collect response or to drive sales or to perform analysis. QR codes are created and printed or otherwise embedded, onto anything from direct mail to postcards, catalogues and more. Email campaigns can also benefit from the use of QR codes.

Just like with the rest of the applications, the codes are used only as an information vector, the actual process of tracking being external to them. A possible tracking scheme might be accomplished using an account with one of the websites dedicated to traffic statistics. *Google Analytics, Clicky or Piwick, BitLy* and many others all provide ways of linking to web pages using special “gateways”. These websites create special links to the user’s content. When someone clicks, they get redirected first to the company host where the request for the target web page is logged. Once this is done, the user is redirected to the actual page they wanted to visit[4]. A QR code fits comfortably into this scheme as one of the most popular options, as described in [9], is creating URL links.

Object hyper-linking and some of its applications

This neologism describes a process by which the Internet is extended to objects and locations in the real world. This is done by attaching object tags with URLs as meta-objects to tangible objects or locations. These object tags can then be read by a wireless mobile device and information about objects and locations retrieved and displayed. In this sense, object hyper-linking is a branch of ambient intelligence.

An important part of this tagging system is the tag itself. There are lots of types of tag, including RFID tags and graphical tags. Their design needs to be able to include lots of information and must be robust enough for the tag to be readable, even when partly obscured or damaged. As an example, graphical tags which are outdoors are exposed to the weather and if they become blurred or partly destroyed, they should still be readable.

Graphical tags have a number of advantages. They are easy to understand and cheap to produce and they can be printed on virtually anything. QR codes are a particularly attractive form of tagging because they are already very widely used, and camera phones can easily read them.

Various examples of the world suggested by object hyper-linking follow.

- Extra information about products in shopping malls. For example, in the meat section, a QR code might take us to the producer’s website and we can find out information about the origin of the animal, its diet and the date it was dispatched. Providing this information is not additional burden on the producer given that they already have it and are required to share it with the wholesale retailer. In the clothes department, a QR code can give us information about the material, especially useful if someone has an allergy or hyper sensitive skin.
- Business cards. A regular business card is definitely not on its way out but a QR code on it will surely enhance it. It might contain the exact same information the card contains but in the VCARD format. Importing the resulting `.vcard` file into

[4]A lot of other websites use a similar ideas for their external links, sometimes the lengthy links in the browser’s URL field can be clearly seen.

the contacts of your mobile phone is therefore as simple as pressing the *Scan* button on your QR code scanner.

- Advertising. Here the possibilities are truly endless. Traditional advertising works on limits. Companies are charged by the airtime, additional advertising space, additional newspaper columns. A QR code fixed to an advertisement means there are literally no limits in the amount of information the ad can include. QR codes can be fixed on bus stops, magazine advertisements, posters, brochures, leaflets, product packaging, labels, bottles... For a company, this is a great way to stay in touch with the clients. By printing QR Codes on products, marketing collateral, advertisements, posters and freebies, interactive communication is instantly activated. News, information, pictures, blogging, marketing, branding can all be aided by this technology.
- Museum exhibitions and concerts. Additional information about the author, the period, the artwork and the materials used, among others, can be instantly obtained using a QR code printed next to the painting or the sculpture when visiting a museum exhibition, including translations into various languages. A QR code on your opera brochure can give you access to the libretto, its history and background, biographies of the singers, etc. The same applies to theatre productions, possibly adding interviews with the actors.
- Cinema advertising. Similar to the previous points with the addition of a possible website of the movie or its trailer.
- Schedules. Whether it is the theatre, the cinema, the bus or train stations or the airport, schedules can be coded into QR codes and imported into your mobile phone in an instant. For this, the VCAL format might be used, about which the QR code specifications say it is ideal. Moreover, once a train ticket is issued, a QR code printed on it can contain the information about the schedule ready to be imported into our calendar.
- Social networking. QR codes can be used to confirm one's assistance to an event organized in a social network, or to indicate one's opinion (Like/Unlike) of a certain topic, website, item of news, etc. It can also be used to confirm friendship or membership of groups.
- Tourism. QR codes can be printed on any object of interest, providing the visitor with relevant information. For example, a guided tour can be organized in this way, in which the visitor knows where to start and is guided using QR codes to the

next place, getting all the information required upon request. In mountain hikes, QR codes can be used to indicate the route to follow, the distance to the next post or to the nearest hut, etc.

- Scavenger hunts. The above idea is perfectly applicable to scavenger hunts. The QR codes can contain the riddles that take the players to the next challenge.
- High-tech romance. Since QR codes can be printed on anything, one can print a QR code on their T-shirt. This code can contain their telephone number, email address, Facebook page, sexual preference or what kind of relationship they are after. It may sound amusing but this surely is a potential usage that should not be ignored light-heartedly.

All these examples are just a fraction of what QR codes can accomplish. Once people are aware of them, their usage is limited only by our imagination.

2.3. The case at hand: QR codes and Android

2.3.1. Why Android?

An important objective of this project is the ease of portability. As many components as possible must be made very easy to re-compile for the target platform. Following is a list of reasons why Android was chosen as the target platform.

A vast number of devices

No other platform offers such a plethora of devices on which to run applications. Android runs on ARMv7 and v8, x86, and MIPS, both 32- and 64-bit editions as of Android 5.0. According to [26], the number of Android devices around the world in 2014 was over 1 billion, giving Android a market share of over 80%. Its direct competitor, *iPhone*, had a market share of 15.4% and Windows Phone a modest 2.8%. Another, no less important reason for this choice is the Android “ecosystem”, described in A.1.3. This of course gives a vast choice of devices and a choice in their use to any developer, which in turn means a variety of deployment options for this project.

A wide variety of applications out of the box

Android itself comes complete with a lot of ready-to-use applications like music player, camera app, a full-featured address book, email client, etc. In addition to this, every manufacturer and, in the case of mobile phones, cell carrier, has their own apps installed on top of these[5]. An Android device is, therefore, fully functional mere seconds after it is activated, which would encourage more people to think about owning one. This in turn means even more possible deployment devices for this project.

A wide availability of additional apps

Android has its own official digital content distribution system called Google Play. According to [19] and [18], in November 2014 there were 1,400,000 apps available, of which 1,200,000 were free and 200,000 were paid. Another bit of statistic, taken from [28], shows over 50 billion downloads and 1,430,000 apps as of January 2015.

A framework which encourages code re-use

Probably the most important feature from a programmer's point of view is the fact that at the level of system organisation, Android has features that allow one application to run parts of another for various side effects. The prerequisite is that the secondary application should declare that it allows to be run. If certain special circumstances are present, whole screens (called *activities* in Android) of one program, either built-in or deliberately installed, can be used at runtime by another one. This is an important feature of the Android framework which allows for massive code re-use and creates potentially unique interactions between programs. It is also at the core of this project.

More about the framework can be found in A.2.

2.3.2. Some QR code-driven applications

A look through *Google Play Store* reveals a number of apps that use QR codes. It is important to note that the focus of these is on capturing, interpretation and creation. This is no surprise as frameworks are usually a behind-the-scenes affair and rarely appear except when described explicitly.

Following is a review of some of the most popular applications that deal with QR codes.

QR Scanner: Free Code Reader by Kaspersky Lab (`http://www.kaspersky.com/qr-scanner`)

[5]Lovingly referred to as "bloatware"

This free application comes from the manufacturers of anti-virus software Kaspersky Labs. It verifies the QR codes which are scanned with it before it allows them to be acted upon. It checks the URLs against a list of malware or phishing websites and warns the user before it allows them to open the links. It also works with WiFi credentials and contact information, as well as images and text messages. In other words, it adds a layer of security to the QR codes by limiting their usage to the ones Kaspersky deem safe.

Figure 2.10.: QR Scanner Logo

QuickMark Lite QR Code Reader by SimpleAct, Inc. (`http://www.quickmark.com.tw`)

This application comes from a Taiwanese company. One of its most outstanding features is the *History* tab, where users can see what they have scanned and saved and when; said tab also provides statistics as to the different types of information found in the scanned QR codes. The application provides insights into how many times the code the user is scanning has been scanned and its rating, provided of course that the codes are configured to be traced, as outlined in section 2.2.2. An additional and very useful feature is that *QuickMark Lite* allows batch scanning of QR codes from various sources such as web pages and loose image files.

Figure 2.11.: QuickMark Lite logo

QR Droid Code Scanner by QRDroid (`http://www.qrdroid.com`)

This application uses the *Zapper*[6] technology, which allows information in QR codes to be used for registering and logging on to websites, checking out items from shopping baskets, paying in adhered establishments, find prices, reviews and directly shop for an item, all these through the *Zapper* infrastructure. Discounts, special rates, additional publicity are also supported using the same framework. *QR Droid* also pays specific

[6] More about can be found at `http://www.zapper.com`

Figure 2.12.: QR Droid logo

attention to sharing via Twitter, Facebook and other social channels. It features various options for decoding from images and websites, as well as generating QR codes.

i-nigma QR & Barcode Scanner by 3GVision (`http://www.3gvision.com/i-nigmahp.html`)

Figure 2.13.: i-nigma logo

Similarly to other apps, *i-nigma* scans other formats in addition to QR codes, such as DataMatrix, supermarket barcodes and many more if the SDK is present. Its multilingual support is a unique feature, none of the other apps reviewed herein contain this in their descriptions. In addition to the most important Western European languages, it supports Russian and Hungarian. Its *GooglePlay* description boasts that *"3GVision*'s mobile scanning SDK is the de facto standard for Japanese headsets"[7]. This app also features optical code scanning from sources other than the camera, and a unique feature is that it can use the input from an MMS as its source.

Comparison

Without considering the previous four apps an exhaustive sample, it can be inferred that applications that use QR codes nowadays are mostly user-oriented, the idea being to focus on functionality and thus to abstract the complexities of the internals of the codes,

[7]Source: `http://www.3gvision.com/QR-Barcode-Reader-SDK.html`

while at the same time adding features such as history, tracking, etc. With the notable exception of *i-nigma*, there is hardly a mention of a framework behind these apps. This should come as no surprise; it can be argued that the proliferation of mobile technology has had this as one of its objectives.

When it comes to security applications, a more representative sample would probably show applications similar to *QR Scanner* by Kaspersky, which combines phishing protection and white-list-like features when a QR code is detected. It is to be expected for security application not to disclose their “internals”, possibly as a security precaution. For this reason, it is exceedingly difficult to provide an estimate of the security frameworks based on QR codes on the market nowadays.

2.4. Development tools: description

2.4.1. Android 4.4

Labelled *KitKat*, Android 4.4 was the current version at the time of the design and implementation of this framework. No specific features to any specific version have been used so it is expected, although not tested, that the GUI part of the framework should function without a need for changes. The API level corresponding to the various releases is 19. A further discussion of Android versions and the various API levels can be found in A.1, more specifically in A.1.1.

2.4.2. Java 7

Android and Java are probably the most frequent combination in the mobile world nowadays. Java has long been the choice for portable applications due to the fact that it is both interpreted and compiled, using the Java VM, the bytecode interpreter and JIT compilation. Its enormous base of built-in functionality makes it one of the catch-all programming languages. It is an important part of the repertoire of any programmer in the current times, which would certainly help to expand the framework in the future.

The reason why Java 7 was chosen was that it provides *Collections* and a number of syntax differences with other versions, like the `switch` operator functioning on an extended set of types, proper and compulsory handling of generics and, as a result of more data types being introduced, more precise typecasting.

2.4.3. Eclipse 3.8 and ADT

A significant part of this framework was written in pure Java, without any thought of deployment on any operating system. This is where *Eclipse* came into play. In later stages, ADT was used to develop most of the non-GUI Android-specific components of the framework. More on this and other ways of developing applications is given in B, specifically B.3.1.

2.4.4. Android Studio 1.2

For the GUI, and following Google's advice, *Android Studio* was used. Being specific to Android, it contains everything needed to develop the GUI components and interaction patterns, program, re-factor, run apps on virtual or physical devices and of course, debug. More information can be found in B.4.

2.4.5. ZXing

A crucial element in any application that uses QR codes is of course the reader and the QR code creator. For Android, there are a number of widely used applications for scanning. In this section we turn our attention to one of them, *Barcode Scanner*, provided by Google and more specifically to the library behind it, *ZXing*[8].

ZXing (pronounced "zebra crossing") is an open-source, multi-format 1D/2D bar-code image processing library implemented in Java. The focus of this project is on using the built-in camera on mobile phones to photograph and decode bar-codes directly on the device, without communicating with a server. Alternatively, ZXing can be used to scan QR codes from a web page. One has to supply the URL for the page and ZXing scans all the codes it can find there and returns the decoded information. ZXing can also be used to generate QR codes. All these uses will be described in the upcoming pages.

Figure 2.14.: ZXing logo

ZXing supports all the formats listed in section 2.1, except for the Shotwell code which requires it own application. A full list of supported codes follows.

[8]Please note that this section, particularly C.2, C.3 and C.4, contains references to many built-in Java classes. Understanding all of them is not required, just when to use them

UPC-A & E	Code 39	ITF	PDF 417
EAN-8 & 13	Code 128	RSS-14	Aztec
Code 39	QR	Data Matrix	Codabar

ZXing contains various modules which can be freely used and are actively developed. The following is a list of ready-to-use parts of the library:

core: The core image decoding library, and test code

javase: J2SE-specific client code

android: Android client, called Barcode Scanner

androidtest: Android test app

android-integration: Supports integration with Barcode Scanner app via Intent

zxingorg: The source behind `http://zxing.org/w`

zxing.appspot.com: The source behind the web-based bar-code generator

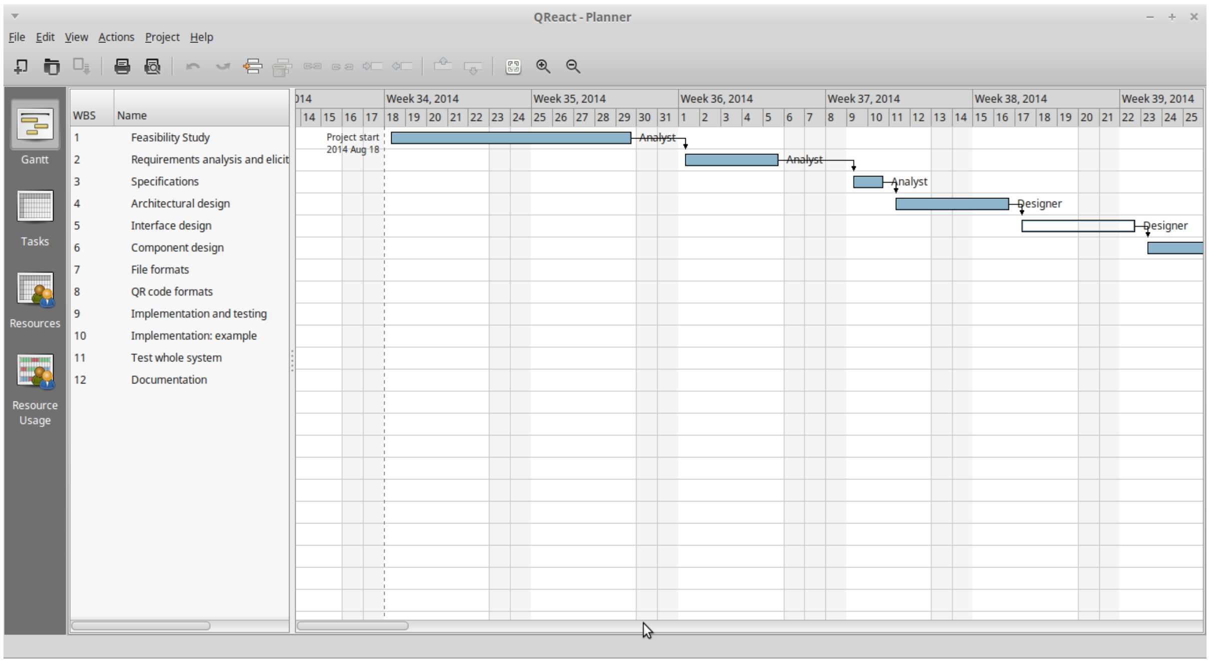
In addition to these, a number of other modules exist, most of which are contributed (third-party) and not maintained by the ZXing team. More information on ZXing can be found in Appendix C.

2.5. Other tools used in this project

2.5.1. Planner (`https://wiki.gnome.org/Apps/Planner/`)

This tool is used to keep track of the budget, time planning and the Gannt diagram of the project. As the website states, "[i]ts goal is to be an easy-to-use no-nonsense cross-platform project management application". Although quite simple, it contains everything needed to manage resources, both human and technical, to keep track of the budget and costs and consequently, calculate overruns if these should occur, and schedule different tasks in various relationships. Excellent capabilities for keeping track of the percentage of tasks done and thus keeping the project on schedule are also provided. A screen shot of its clean and simple interface is shown in figure 2.15.

This basic application proved more than sufficient for the needs of this project, particularly when tackling the somewhat daunting tasks of costs and budgeting.



WBS	Name
1	Feasibility Study
2	Requirements analysis and elicit
3	Specifications
4	Architectural design
5	Interface design
6	Component design
7	File formats
8	QR code formats
9	Implementation and testing
10	Implementation: example
11	Test whole system
12	Documentation

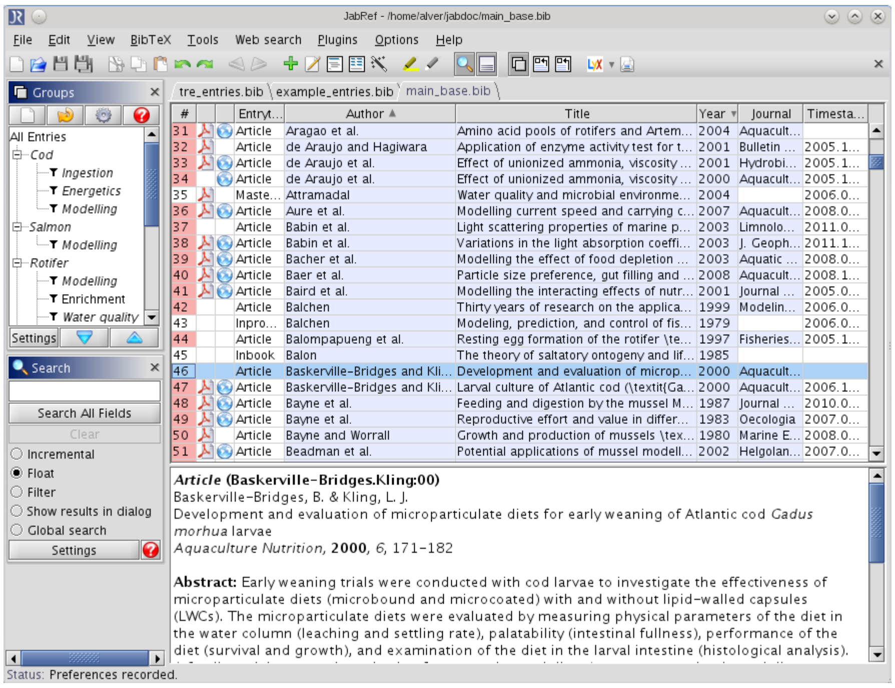
Figure 2.15.: Planner screen shot

2.5.2. JabRef (`http://jabref.sourceforge.net/`)

Any project of this size requires quite a bit of reading and consequently, a way to keep track of the references. JabRef is a bibliography reference manager. Written in Java and therefore being completely cross-platform, it is compatible with BibTex, the default LaTeX (section 2.5.3) bibliography database-like file format, and with many others. A screen shot, taken from the website, is shown in figure 2.16. The basic version features, among others, detailed editing of all entries, the possibility of classifying them according to multiple criteria, support for translations, search functions, both within the database and on academic websites like Google Scholar, Medline, arXiv, etc. It can be infinitely customizable through plug-ins.

2.5.3. LaTeX and LyX (`www.ctan.org` and `www.lyx.org`)

As a document-preparation system, LaTeX has long been favoured among scientists and professionals as the typesetting system *par excellence*. Its modular design together with the vast amount of additional functionality available completely free of charge, its cross-platform nature, the vast online knowledge base and of course, the beautifully typeset documents, unimaginable with anything else, are just some of the reasons for this. LyX



#			Entryt...	Author ▲	Title	Year ▼	Journal	Timesta...
31	🗎	🌐	Article	Aragao et al.	Amino acid pools of rotifers and Artem...	2004	Aquacult...	
32	🗎		Article	de Araujo and Hagiwara	Application of enzyme activity test for t...	2001	Bulletin ...	2005.1...
33	🗎	🌐	Article	de Araujo et al.	Effect of unionized ammonia, viscosity ...	2001	Hydrobi...	2005.1...
34		🌐	Article	de Araujo et al.	Effect of unionized ammonia, viscosity ...	2000	Aquacult...	2005.1...
35	🗎		Maste...	Attramadal	Water quality and microbial environme...	2004		2006.0...
36	🗎	🌐	Article	Aure et al.	Modelling current speed and carrying c...	2007	Aquacult...	2008.0...
37			Article	Babin et al.	Light scattering properties of marine p...	2003	Limnolo...	2011.0...
38	🗎	🌐	Article	Babin et al.	Variations in the light absorption coeffi...	2003	J. Geoph...	2011.1...
39	🗎	🌐	Article	Bacher et al.	Modelling the effect of food depletion ...	2003	Aquatic ...	2008.0...
40	🗎	🌐	Article	Baer et al.	Particle size preference, gut filling and ...	2008	Aquacult...	2008.1...
41	🗎	🌐	Article	Baird et al.	Modelling the interacting effects of nutr...	2001	Journal ...	2005.0...
42			Article	Balchen	Thirty years of research on the applica...	1999	Modelin...	2006.0...
43			Inpro...	Balchen	Modeling, prediction, and control of fis...	1979		2006.0...
44			Article	Balompapueng et al.	Resting egg formation of the rotifer \te...	1997	Fisheries...	2005.1...
45			Inbook	Balon	The theory of saltatory ontogeny and lif...	1985		
46			Article	Baskerville-Bridges and Kli...	Development and evaluation of microp...	2000	Aquacult...	
47	🗎	🌐	Article	Baskerville-Bridges and Kli...	Larval culture of Atlantic cod (\textit{Ga...	2000	Aquacult...	2006.1...
48	🗎	🌐	Article	Bayne et al.	Feeding and digestion by the mussel M...	1987	Journal ...	2010.0...
49	🗎	🌐	Article	Bayne et al.	Reproductive effort and value in differ...	1983	Oecologia	2007.0...
50	🗎		Article	Bayne and Worrall	Growth and production of mussels \tex...	1980	Marine E...	2008.0...
51	🗎	🌐	Article	Beadman et al.	Potential applications of mussel modell...	2002	Helgolan...	2007.0...

***Article* (Baskerville-Bridges.Kling:00)**
Baskerville-Bridges, B. & Kling, L. J.
Development and evaluation of microparticulate diets for early weaning of Atlantic cod *Gadus morhua* larvae
Aquaculture Nutrition, **2000**, *6*, 171-182

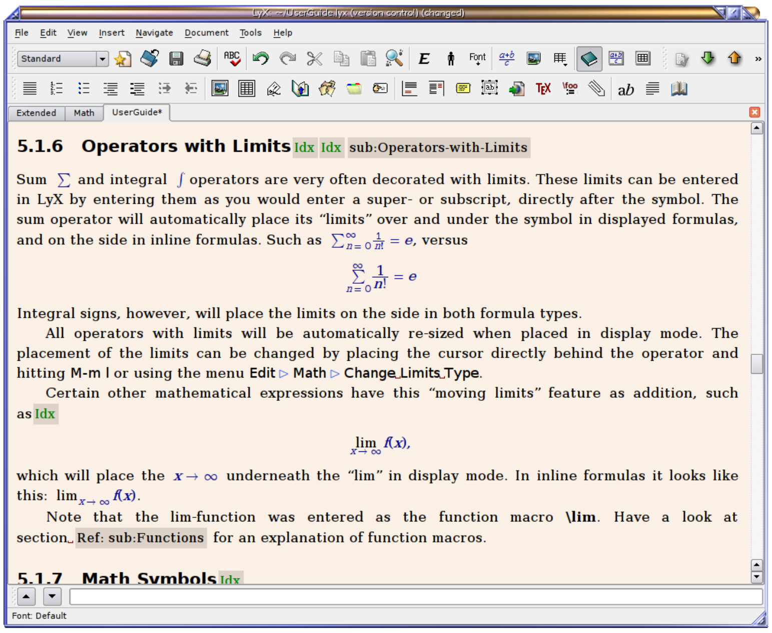
Abstract: Early weaning trials were conducted with cod larvae to investigate the effectiveness of microparticulate diets (microbound and microcoated) with and without lipid-walled capsules (LWCs). The microparticulate diets were evaluated by measuring physical parameters of the diet in the water column (leaching and settling rate), palatability (intestinal fullness), performance of the diet (survival and growth), and examination of the diet in the larval intestine (histological analysis).

Figure 2.16.: JabRef screen shot

(figure 2.17, taken from its website) was chosen as a powerful and capable front-end due to its familiar design and ease of use. It is tremendous help in dealing with multiple documents, document styles, graphics, numbering, exporting into other formats, glossary and index features and so many more things LaTeX users have to face on a daily basis. A great aid is of course the spell-check and the built-in installer for style files. Its real strengths, however, are its mathematics mode, its pipe-like connection with CAS and the possibility of directly programming in-line graphics.

2.5.4. TaskCoach (`http://taskcoach.org`)

Last but definitely not least, a project simply cannot go without an application to handle various unsorted to-do lists and items. It is true that Eclipse has support for to-do lists inside any given project but those cover mostly the programming side of the question. TaskCoach is much more than a to-do list manager. Written in Python and totally cross-platform (it even supports *iOS*, for a small fee), it features support for multiple projects, scheduling and time tracking options, date-driven tick-off to do lists including attachments, and a lot more. Figure 2.18 shows a screen shot, taken from its website.



5.1.6 Operators with Limits Idx Idx sub:Operators-with-Limits

Sum $\sum$ and integral $\int$ operators are very often decorated with limits. These limits can be entered in LyX by entering them as you would enter a super- or subscript, directly after the symbol. The sum operator will automatically place its "limits" over and under the symbol in displayed formulas, and on the side in inline formulas. Such as $\sum_{n=0}^{\infty}\frac{1}{n!}=e$, versus

$$\sum_{n=0}^{\infty}\frac{1}{n!}=e$$

Integral signs, however, will place the limits on the side in both formula types.

All operators with limits will be automatically re-sized when placed in display mode. The placement of the limits can be changed by placing the cursor directly behind the operator and hitting M-m l or using the menu Edit ▷ Math ▷ Change_Limits_Type.

Certain other mathematical expressions have this "moving limits" feature as addition, such as Idx

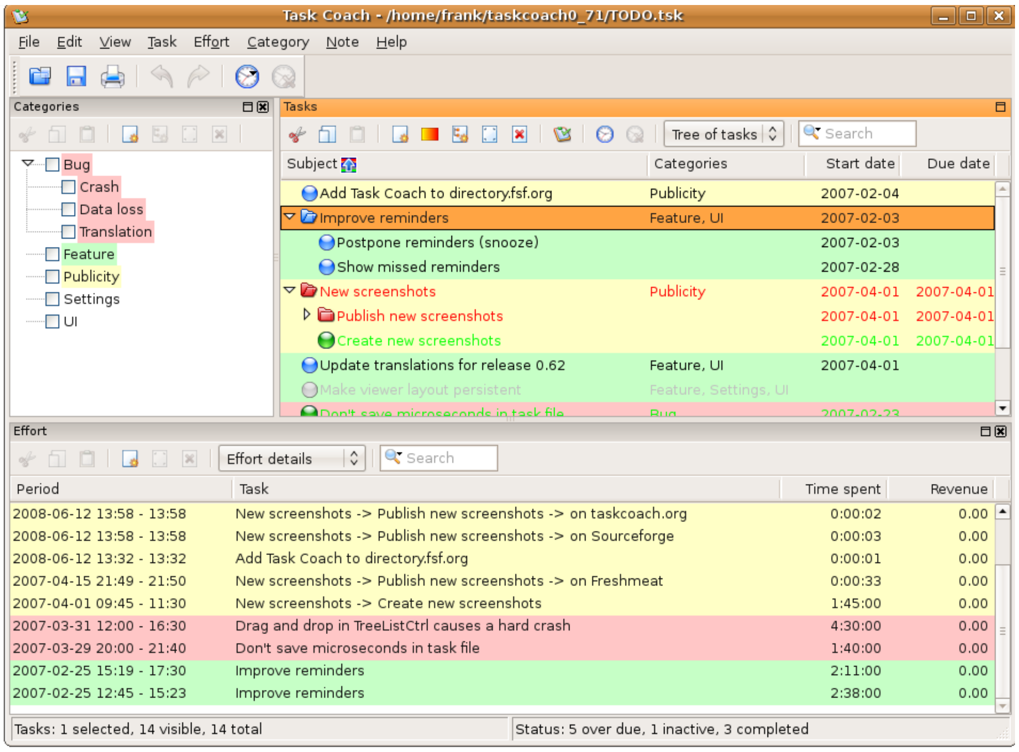
$$\lim_{x\to\infty} f(x),$$

which will place the $x\to\infty$ underneath the "lim" in display mode. In inline formulas it looks like this: $\lim_{x\to\infty} f(x)$.

Note that the lim-function was entered as the function macro \lim. Have a look at section Ref: sub:Functions for an explanation of function macros.

5.1.7 Math Symbols Idx

Figure 2.17.: LyX screen shot


Task Coach - /home/frank/taskcoach0_71/TODO.tsk
File Edit View Task Effort Category Note Help
Categories
Tasks
Tree of tasks
Search
Effort
Effort details
Search

Categories:
- Bug
  - Crash
  - Data loss
  - Translation
- Feature
- Publicity
- Settings
- UI

Subject	Categories	Start date	Due date
Add Task Coach to directory.fsf.org	Publicity	2007-02-04	
Improve reminders	Feature, UI	2007-02-03	
Postpone reminders (snooze)		2007-02-03	
Show missed reminders		2007-02-28	
New screenshots	Publicity	2007-04-01	2007-04-01
Publish new screenshots		2007-04-01	2007-04-01
Create new screenshots		2007-04-01	2007-04-01
Update translations for release 0.62	Feature, UI	2007-04-01	
Make viewer layout persistent	Feature, Settings, UI		
Don't save microseconds in task file	Bug	2007-02-23	

Period	Task	Time spent	Revenue
2008-06-12 13:58 - 13:58	New screenshots -> Publish new screenshots -> on taskcoach.org	0:00:02	0.00
2008-06-12 13:58 - 13:58	New screenshots -> Publish new screenshots -> on Sourceforge	0:00:03	0.00
2008-06-12 13:32 - 13:32	Add Task Coach to directory.fsf.org	0:00:01	0.00
2007-04-15 21:49 - 21:50	New screenshots -> Publish new screenshots -> on Freshmeat	0:00:33	0.00
2007-04-01 09:45 - 11:30	New screenshots -> Create new screenshots	1:45:00	0.00
2007-03-31 12:00 - 16:30	Drag and drop in TreeListCtrl causes a hard crash	4:30:00	0.00
2007-03-29 20:00 - 21:40	Don't save microseconds in task file	1:40:00	0.00
2007-02-25 15:19 - 17:30	Improve reminders	2:11:00	0.00
2007-02-25 12:45 - 15:23	Improve reminders	2:38:00	0.00

Tasks: 1 selected, 14 visible, 14 total

Status: 5 over due, 1 inactive, 3 completed

Figure 2.18.: TaskCoach screen shot

3. Analysis, design and implementation

"It is easier to change the specification to fit the program than vice versa"
Brian Kerninghan

This chapter deals with the processes of analysis of requirements, design of the framework and implementation and programming. A specially important feature is the syntax of the QR codes which needs to be clearly defined and strictly controlled. A brief overview of the tests performed and their results is presented at the end.

3.1. Analysis

3.1.1. User requirements

This section lists the user requirements, both *functional* and *restriction.* To facilitate reading these, tabular format has been chosen, in which the following fields are present:

name is a descriptive name of the requirement

id is a unique identifier, which in this section is of the format **URYXX. Y** is substituted for **F** if the requirement is **functional** and **R** if it is a restriction requirement. **XX** is a two-digit number beginning with **01**.

priority defines how important satisfying this requirement is to the overall project progress. It can take the values *time-critical, medium* or *low.* This field helps schedule the development process

type defines whether the requirement is *essential* or *desirable* to the overall functioning of the project

verifiable states how easy it is to ensure that the requisite is included in the project. It can have the values of *easily, with difficulties* or *not*

stable answers the question of how likely it is that the requirement might change over the development of the framework. A value of *yes* indicates the requirement will likely not change, a value of *no* means it most probably will

description is a detailed description of the requirement

3.1.1.1. Functional requirements

"Blind" creation of QR codes		**URF01**	
priority	time-critical	*type*	essential
verifiable?	easily	*stable?*	yes
description	The user must be presented with a way to create QR codes without being aware of the current setup of the framework		

Table 3.1.: User requirement URF01

Control over QR formats		**URF02**	
priority	time-critical	*type*	essential
verifiable?	easily	*stable?*	yes
description	The user must have a systematic way to control every aspect of the QR code format		

Table 3.2.: User requirement URF02

Dynamic control over configuration		**URF03**	
priority	time-critical	*type*	essential
verifiable?	easily	*stable?*	yes
description	The user must have control over how the framework is configured in order to embed it into a complete system. If any changes should occur, they must be updated immediately.		

Table 3.3.: User requirement URF03

Control over action on QR codes		**URF04**	
priority	time-critical	*type*	essential
verifiable?	with difficulties	*stable?*	yes
description	The user must be able to define their own ways of interacting with the host system		

Table 3.4.: User requirement URF04

Access to usability and usage statistics		**URF05**	
priority	medium	*type*	desirable
verifiable?	easily	*stable?*	yes
description	The user must have easy access to what QR codes where scanned and when, as well as to the usable codes, at any moment		

Table 3.5.: User requirement URF05

Built-in syntax and semantic checks without acting on QR codes		**URF06**	
priority	medium	*type*	desirable
verifiable?	easily	*stable?*	yes
description	The user must know whether a QR code can be acted upon without actually executing it, at any moment		

Table 3.6.: User requirement URF06

Built-in actions on QR codes		**URF07**	
priority	time-critical	*type*	essential
verifiable?	easily	*stable?*	yes
description	The user must have clearly defined actions when a QR code is presented		

Table 3.7.: User requirement URF07

Configuring the framework through QR codes		**URF08**	
priority	time-critical	*type*	essential
verifiable?	easily	*stable?*	yes
description	The user must be able to configure the framework using special QR codes		

Table 3.8.: User requirement URF08

Information about configured QR codes		**URF09**	
priority	medium	*type*	essential
verifiable?	easily	*stable?*	yes
description	The user must be able to access all information regarding a particular configured QR code		

Table 3.9.: User requirement URF09

Two-way communication with the host application		**URF10**	
priority	medium	*type*	essential
verifiable?	easily	*stable?*	yes
description	The user must be able to access information exchanged between host app and framework		

Table 3.10.: User requirement URF10

3.1.1.2. Restriction requirements

Impossibility to act on malformed QR codes		**URR01**	
priority	time-critical	*type*	essential
verifiable?	easily	*stable?*	yes
description	The user must not be allowed to act on syntactically incorrect codes		

Table 3.11.: User requirement URR01

Impossibility to create non-standard QReact codes		**URR02**	
priority	time-critical	*type*	essential
verifiable?	easily	*stable?*	yes
description	The user must not be allowed to create syntactically incorrect codes		

Table 3.12.: User requirement URR02

Impossibility to create QReact codes directly		**URR03**	
priority	time-critical	*type*	essential
verifiable?	easily	*stable?*	yes
description	The user must not be allowed to create QR codes directly; this must be done using the built-in functions		

Table 3.13.: User requirement URR03

Impossibility to bypass the framework logic at run-time		**URR04**	
priority	time-critical	*type*	essential
verifiable?	easily	*stable?*	yes
description	The user must not be allowed to interfere with the framework functions at run-time. These include awareness of which QR codes can be acted upon.		

Table 3.14.: User requirement URR04

3.1.2. Software requirements

This section lists the software requirements, both *functional* and *non-functional.* The format followed is almost identical to the one used in section 3.1.1, except for an additional field:

dependencies lists the user requirement code(s) associated with the software requirement

The identifiers also follow a different format. In the upcoming tables, it is **SRYXX,** where **Y** is substituted for **F** in the case of a functional requirement and **NF** if it is non-functional. In either case, **XX** is a number starting from **01**.

Interface to create valid QR codes		**SRNF01**	
priority	time-critical	*type*	essential
verifiable?	easily	*stable?*	yes
description	The framework must provide a function to generate valid QR codes		
dependencies	**URF01, URR03**		

Table 3.15.: Software requirement SRNF01

Interface to validate QR codes		**SRNF02**	
priority	time-critical	*type*	essential
verifiable?	easily	*stable?*	yes
description	The framework must provide a set of functions to validate QR codes at various depths		
dependencies	**URR01**		

Table 3.16.: Software requirement SRNF02

Usage statistics		**SRNF03**	
priority	time-critical	*type*	essential
verifiable?	easily	*stable?*	no
description	The framework must keep track of: • all QR codes that can be acted upon *and* • all QR codes that have been scanned, regardless of whether they have been handled or not		
dependencies	**URF05, URR04**		

Table 3.17.: Software requirement SRNF03

Dynamic configuration		**SRNF04**	
priority	time-critical	*type*	essential
verifiable?	easily	*stable?*	no
description	The framework must be configured dynamically by the host application		
dependencies	**URF03, URF10, URR04**		

Table 3.18.: Software requirement SRNF04

Information about configured QR codes		**SRNF05**	
priority	time-critical	*type*	essential
verifiable?	easily	*stable?*	no
description	The framework must provide information about configured QR codes		
dependencies	**URF09, URF10**		

Table 3.19.: Software requirement SRNF05

Access to information about configured QR codes		**SRF06**	
priority	time-critical	*type*	essential
verifiable?	easily	*stable?*	no
description	When the host system requires it, a list containing every aspect of a QR code will be provided; also, QR code definitions will be converted into into various exchange formats		
dependencies	**URF09, URF10**		

Table 3.20.: Software requirement SRNF06

Access to dynamic configuration		**SRF07**	
priority	time-critical	*type*	essential
verifiable?	easily	*stable?*	no
description	When so prompted by the host application, the relevant parts of the framework will re-configure themselves.		
dependencies	**URF03, URF10, URR04**		

Table 3.21.: Software requirement SRF07

YAML input/output		**SRF08**	
priority	time-critical	*type*	essential
verifiable?	easily	*stable?*	no
description	The framework must provide functions which: • handle file operations in a YAML-like format • handle exceptions that arise during these • provide conversion functions (list to YAML and vice versa)		
dependencies	**URF03, URF05, URF10**		

Table 3.22.: Software requirement SRF08

Change in QR code formats		**SRF09**	
priority	time-critical	*type*	essential
verifiable?	easily	*stable?*	yes
description	When a change in the QR code formats is called for, their definitions will be altered accordingly in *only one* file. Every aspect of the codes can be changed		
dependencies	**URF02, URR01, URR02**		

Table 3.23.: Software requirement SRF09

Semantic validation of QR codes		**SRNF10**	
priority	medium	*type*	essential
verifiable?	easily	*stable?*	yes
description	As a precaution and for validation purposes, a simulation of handling a ready-to-handle QR code will be offered, with the error messages produced.		
dependencies	**URF06, URR01, URR04**		

Table 3.24.: Software requirement SRNF10

Validation procedure for QR codes		**SRF11**	
priority	medium	*type*	essential
verifiable?	easily	*stable?*	yes
description	• Parse the presented QR code, going through all the necessary steps as if it was being handled • if an error is present, the error message is returned • in no case is any exception raised		
dependencies	**URF06, URR01, URR04**		

Table 3.25.: Software requirement SRF11

Built-in actions for QR codes		**SRNF12**	
priority	medium	*type*	essential
verifiable?	easily	*stable?*	yes
description	Four default actions will be presented for every QR code; its contents will select one of them and the optional parameters needed for its handling. It should be easy to alter these by adding more definitions and behaviour in the appropriate places in the code.		
dependencies	**URF07, URF04, URR02, URR04**		

Table 3.26.: Software requirement SRF12

Procedure for executing built-in actions for QR codes		**SRF13**	
priority	medium	*type*	essential
verifiable?	easily	*stable?*	yes
description	For every defined command do: • check syntax. This varies depending on the command • check semantics • if everything is alright, execute command and return with a non-error status • otherwise, raise the appropriate exception		
dependencies	**URF07, URF04, URR02, URR04**		

Table 3.27.: Software requirement SRF13

Procedure for extending built-in actions for QR codes		**SRF14**	
priority	medium	*type*	essential
verifiable?	easily	*stable?*	yes
description	• define command code in the appropriate component • append command semantics to the appropriate component • test using the dry-run function		
dependencies	**URF07, URF04, URR02, URR04**		

Table 3.28.: Software requirement SRF14

Configuring the framework using QR codes		**SRNF15**	
priority	time-critical	*type*	essential
verifiable?	easily	*stable?*	yes
description	The framework allows basic configuration functionality called on by QR codes. These, by default, are editing, adding and erasing a QR definition.		
dependencies	**URF08, URR04**		

Table 3.29.: Software requirement SRNF15

Procedure for adding definitions using QR codes		**SRF16**	
priority	time-critical	*type*	essential
verifiable?	easily	*stable?*	yes
description	• the default `ADD` command takes an ID, a program name and, optionally, parameters to append to the list of definitions. • when this command is presented to the QR parsing module, it is validated and if the same definition does not exist, it is added to the list of QR definitions		
dependencies	**URF08, URR04**		

Table 3.30.: Software requirement SRF16

Procedure for editing definitions using QR codes		**SRF17**	
priority	time-critical	*type*	essential
verifiable?	easily	*stable?*	yes
description	• the default `EDIT` command takes an ID, a program name and, optionally, parameters to append to the list of definitions. Its syntax is essentially the same as the `ADD` command • when this command is presented to the QR parsing module, it is validated • if the same definition does not exist, an exception is raised • if it does, the new program and parameters overwrite the old definition. The QR ID is kept as is.		
dependencies	**URF08, URR04**		

Table 3.31.: Software requirement SRF17

Procedure for removing definitions using QR codes		**SRF18**	
priority	time-critical	*type*	essential
verifiable?	easily	*stable?*	yes
description	• the default `DELETE` command takes only an ID • when this command is presented to the QR parsing module, it is validated • if the definition does not exist, an exception is raised • if it does, it is removed from the list. This operation cannot be undone.		
dependencies	**URF08, URR04**		

Table 3.32.: Software requirement SRF18

3.1.3. Traceability matrix

Having put together the requirements that must be satisfied, it is important to make sure that every user requirement is covered by at least one software requirement. For this, a traceability matrix is used. Its vertical axis contains the user requirements. The horizontal axis is left to the software requirements. When these intersect, the symbol ✓ is used. Table 3.33 shows the traceability matrix. It is important that every user requirement is covered by at least one software requirement, otherwise the design will be incorrect. In the traceability matrix this means that no row should be completely blank. If this were to happen, the design should be revised.

3.2. Design

This section presents the design phase of the development of the framework. Architectural aspects are considered, as well as code and file formats.

3.2.1. Architecture: design considerations

The design process followed in *QReact* is different from the normal process to follow when designing an application. A framework necessarily must be built into a host, to which it provides basic functionality. The host, on the other hand, needs to provide some configuration information for the framework to work properly. During the design phase, the focus must be on how to interact with the host system. Failure to do this will make it look like certain features are left as "undefined" or not clearly defined, on both sides.

3.2.1.1. *QReact* and its environment

Figure 3.1 represents the relationship between *QReact* and its environment, the focus being on its interaction with the host or driver application. The arrows indicate the direction in which the data moves.

	SRNF01	SRNF02	SRNF03	SRNF04	SRNF05	SRF06	SRF07	SRF08	SRF09	SRNF10	SRF11	SRNF12	SRF13	SRF14	SRNF15	SRF16	SRN17	SRN18
URF01	✓					✓												
URF02									✓									
URF03				✓				✓										
URF04							✓					✓	✓	✓				
URF05			✓					✓										
URF06										✓	✓							
URF07												✓	✓	✓				
URF08															✓	✓	✓	✓
URF09					✓	✓												
URF10				✓	✓	✓	✓	✓										
URR01		✓							✓	✓	✓							
URR02									✓			✓	✓	✓				
URR03	✓																	
URR04			✓	✓			✓			✓	✓	✓	✓	✓	✓	✓	✓	✓

Table 3.33.: Traceability matrix between user and software requirements

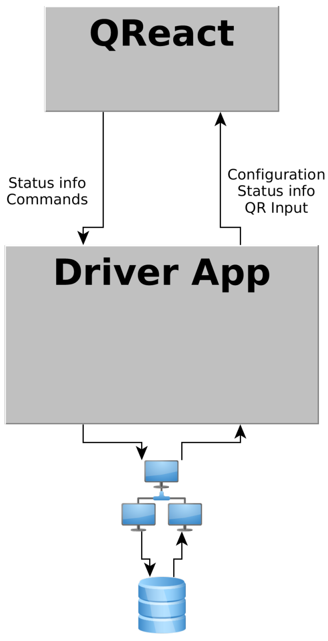


Figure 3.1.: *QReact* and its environment

configuration refers to information regarding the configured QR codes, config files and their location, etc.

status info refers to changes in the the configuration files, or applications becoming temporarily unavailable. When *QReact* sends status info, it usually is about the configured QR codes or the ones which have been scanned

QR input is the input *QReact* needs to work with

commands is basically “QR code accepted or rejected” or error messages

3.2.1.2. *QReact*'s building blocks

It is important to note that *QReact* does not distinguish between users and administrators. The framework's design is egalitarian in that it can be used for access control and also for defining the QR codes. There is no GUI as such; an example driver application is included with the sole purpose of verifying the design and the correctness of the design. Frequently, communication with an external application involves exception handling. In addition, almost complete freedom to redefine behaviour and QR code formats is called for. Last but not least, it is desirable to make the framework easy to embed and to port to other operating systems.

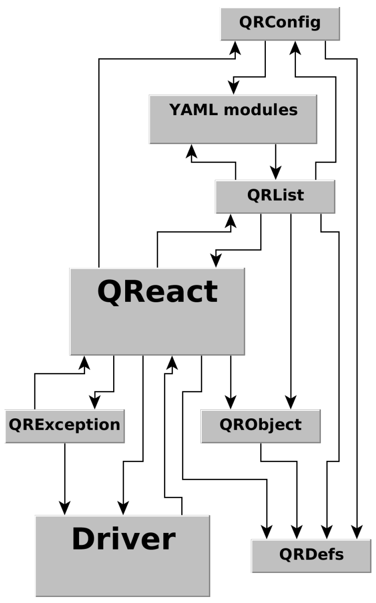


Figure 3.2.: Component relationship diagram

Taking all the above mentioned features into consideration, a strong modular design was opted for. Figure 3.2 shows the relationship between the various classes that compose *QReact,* using their names from the implementation phase. Once again, the arrows indicate the direction in which the data moves. The example host or driver application is included for the sake of completeness.

3.2.1.3. Component specification

This section provides a detailed description of all components which have been considered in the design phase. In order to facilitate their link to the design requirements, tables have been used, whose fields have the following format:

name is a descriptive name of the component

ID is the component's unique identifier. It has the format **COMXX,** where **XX** is the component number

purpose is a short description of what the component does in the framework

description is a list of the functions the component fulfils

requirements shows which user requirements the component meets. A correct design will meet all user requirements.

It is important to note that the components' names are not the same as the classes presented in figure 3.2. This is due to the fact that a given class implements more than one component and conversely, the functionality of a given component is spread among various classes.

name	**QR code information manager**	*ID*	**COM01**
purpose	To provide convenient access to all information relevant to a QR code definition		
description	• Provides sufficient fields, both optional and required, to store information about a QR definition from different sources • Provides methods to modify, access, copy, and export said information		
requirements	**URF04, URF09, URF10**		

Table 3.34.: QR code information manager component

name	**QR list manager**	*ID*	**COM02**
purpose	To provide access to a (in principle unlimited) number of QR definitions (via **COM01**).		
description	• Provides access to a simply linked list of all definitions with methods to add, edit, remove, query and access individual elements • Can receive information from different sources, both from *QReact* and from the OS, both one at a time and in bulk		
requirements	**URF04, URF09, URF10, URR04**		

Table 3.35.: QR list manager component

name	**YAML format manager**	*ID*	**COM03**
purpose	To define a minimalist YAML format used to store and exchange information consistently		
description	Defines a minimalist subset of YAML to be used throughout *QReact*		
requirements	**URF05, URF09,URF10, URR04**		

Table 3.36.: YAML format manager component

name	**File input/output manager**	*ID*	**COM04**
purpose	To provide high-level I/O access to various YAML-formatted files		
description	• Can read and write a variety of YAML files, formatted per the rules set by **COM03**. • Has access to OS lower-level file-handling functions • Handles I/O exceptions appropriately		
requirements	**URF05, URF09,URF10, URR04**		

Table 3.37.: File input/output manager component

name	**QR format manager**	*ID*	**COM05**
purpose	To provide modifiable syntactic rules for the format of the QR codes in use by *QReact*		
description	• Provides clearly defined syntactic rules to be followed when building the QR codes • Provides facilities for these rules to be modified		
requirements	**URF04, URF07, URF08, URR04**		

Table 3.38.: QR format manager component

name	**QR syntax checker**	*ID*	**COM06**
purpose	To ensure that the QR codes in use are syntactically correct		
description	• Checks that every QR code which passes through the system conforms to the specifications set forth by **COM05** • Provides methods to check the formats in varying level of complexity • Provides specific string-handling capabilities		
requirements	**URF06, URR01, URR02, URR03**		

Table 3.39.: QR syntax checker component

name	**Configuration manager**	*ID*	**COM07**
purpose	To provide the location of all configuration and interaction files systematically and adapting to changes		
description	• Keeps track of where the configuration and interaction files are stored • Can interact with the host application if these need changing		
requirements	**URF03, URF04, URF10, URR04**		

Table 3.40.: Configuration manager component

name	**Exception manager**	*ID*	**COM08**
purpose	To provide ways to handle informative exceptions in case any irreparable errors occur		
description	• Defines exceptions with specific and informative contents • Can interact with the host application as well as the rest of the framework's components		
requirements	**URF10, URR01, URR02, URR03, URR04**		

Table 3.41.: Exception manager component

name	**QR parse manager**	*ID*	**COM09**
purpose	To parse QR codes and act accordingly		
description	• Uses established QR syntax • Can inform of errors without interrupting the flow of program logic • Can decide what to do with each QR code it receives as a parameter • Can be upgraded to include more commands		
requirements	**URF06, URF07, URF10, URR04**		

Table 3.42.: QR parse manager component

name	**Encryption manager**	*ID*	**COM10**
purpose	To provide an encryption layer to the QR definitions		
description	• To obfuscate the contents of the codes in use • To decipher the contents of the codes in use • Can be changed for a stronger algorithm or a dynamic choice of algorithms		
requirements	**URF07, URF10, URR02, URR04**		

Table 3.43.: Encryption manager component

name	**Book-keeping module**	*ID*	**COM11**
purpose	To keep track of unique identifiers, commands, available, executable, and attempted QR codes		
description	• Uses established QR syntax • Links with all of *QReact* and the host application • Responsible for the definition and proper handling of commands, as well as for every change in these • Responsible for providing usage statistics; for this it communicates with system functions to get timestamps • Responsible for dynamically keeping track of which programs are available, thereby determining which QR codes can be executed • Can be changed to include different usage statistics		
requirements	**URF01, URF02, URF03, URF04, URF05, URF07, URF10, URR01, URR02, URR03, URR04**		

Table 3.44.: Book-keeping component

3.2.1.4. Traceability matrix between user requirements and components

Just as with the software requirements, a traceability matrix is presented which explains the relationship between the components and the user requirements. It is important to ensure that every requirement is covered by at least one software component. If not, the design is incomplete and must be re-done. Table 3.45 contains the relevant data.

	COM01	COM02	COM03	COM04	COM05	COM06	COM07	COM08	COM09	COM10	COM11
URF01											✓
URF02											✓
URF03							✓				✓
URF04	✓	✓			✓		✓				✓
URF05			✓	✓							✓
URF06						✓			✓		
URF07					✓				✓	✓	✓
URF08					✓						
URF09	✓	✓	✓	✓							
URF10	✓	✓	✓	✓			✓	✓	✓	✓	✓
URR01						✓		✓			✓
URR02						✓		✓		✓	✓
URR03						✓		✓			✓
URR04		✓	✓	✓	✓		✓	✓	✓	✓	✓

Table 3.45.: Traceability matrix of components and user requirements

3.2.2. Design of the QR format

The driving force behind this framework is the QR codes. It is imperative, therefore, that their format be clearly defined. In addition to his, it is equally important that a high level of customizability be permitted.

As with many other formats, *QReact* uses a structure that consists of *headers* and a *payload*. Separators as necessary and in addition to this, any QR code can in principle involve a number of different actions and parameters. Table 3.46 contains the control sequences used in the default QR code formats, whereas table 3.47 contains their exact order. Finally, table 3.48 defines the built-in commands.

Additionally, a layer of encryption is included. In its default version, it is a very simple ROT 25 substitution cipher.

parameter	*length*	*value*	*remarks*
`start_sq`	5	`!#RCT`	used to indicate the start of the QR code
`cmd_delim`	2	`#!`	separates the various parts of the command. Must be used also after the last parameter
`cmd_id`	4	`N/A`	indicates what the QR code is to do. By default, there are four commands
`payload`	`N/A`	`N/A`	the actual content of the QR code, i.e. the parameters of the command.
`payload_delim`	1	,	optional. If parts of the payload need to be separated, this is the symbol to use
`end_sq`	1	`!`	every valid QR code must end with this symbol

Table 3.46.: Default *QReact* control sequences

`start_sq`	`cmd_delim`	`cmd_id`	`cmd_delim`	`code_id`	`payload`	`cmd_delim`	`end_sq`

Table 3.47.: Default *QReact* format

command	*code*	*remarks*
`ADD`	`0981`	adds a definition of a QR code to the system
`EDIT`	`0341`	overwrites an already added definition
`REMOVE`	`7647`	erases a definition
`PASS`	`7614`	passes its parameters to an application or activity to be further processed

Table 3.48.: Default commands and their meanings

There is no need to use this format directly, the *QRDefs* component contains a method which forms syntactically valid QR codes.

3.2.3. Design of the file exchange formats[1]

QR code definitions need to be stored in order for the framework to know what codes are defined. Even though *QReact* does not deal with this directly, it uses a simple file format to store the QR definitions and to provide the host system with a list of QR codes which

[1]**Note**: while syntactically valid, the examples listed in this section are **not compatible** with each other and have **never** been used in actual tests.

have been and can be used. Said format is a minimal subset of YAML, defined by the following rules:

1. the storage is of simple `option:value` pairs, each one on its own new line.
2. only one `:` symbol is permitted per line.
3. the individual items will be separated by the `---` sequence on a new line on its own.
4. no separator will be placed after the last line

Below are the options that have been considered basic and indispensable as far as the formats are concerned.

Table 3.49 contains the definitions used to store the configured QR codes. It is very close in content to table 3.51 in section 3.3.1 and to gain a better understanding of these formats, both must be considered in conjunction. To clarify things, listing 3.1 shows a sample configuration file containing three QR codes.

Table 3.50 shows a typical file containing the QR codes that were used, together with a time-stamp (identical to the one produced by the Linux `date` built-in function) recording the moment of usage.

Finally, listing 3.2 shows the IDs of the usable QR codes. A code is deemed usable when

1. it has been properly configured **and**
2. the program to be called when said code is scanned is present in the host system

Whether a code is usable or not is a situation that can vary with time.

name	*usage*
`id`	unique identifier of the QR code
`name`	descriptive name given by the user
`prog`	activity or program associated with the QR code
`tag`	further information associated with the code, if desired
`params`	number of parameters the QR code contains

Table 3.49.: Default options for configured QR codes

```
id: 0x9322
prog: com.google.android.webview.VIEW
```

```
tag: 5
name: Webview
params: 1
---
id: 0x9974
prog: com.myproject.SEND
tag: timestamp and code length
name: Transfer
params: 2
---
id: 0x4718
prog: com.atanas.QReact
tag: password and my secret code
name: Password to open the door
params: 2
```

Listing 3.1: An example of a QReact configuration file

```
id: 0x0fee30234
timestamp: Wed May 13 18:45:03 CEST 2015
---
id: 0x0fee94134
timestamp: Fri May 22 14:32:25 CEST 2015
```

Table 3.50.: Used QR codes format

```
usable: 0x74899
---
usable: 0x14667
---
usable: 0xe8yt4
---
usable: 0xuni34
```

Listing 3.2: An example of a *QReact* file listing usable codes

3.3. Implementation

This section describes the components which were programmed as a part of *QReact.* For each component, a brief description and the most relevant usability notes are presented.

Section 3.3.8 provides clues as to which modules will need to be adapted if the framework needs to be embedded or the formats changed. At the end, a class diagram which illustrates graphically the relationship between the various components is presented.

3.3.1. QRObject

In many ways, *QRObject* is at the core of *QReact*. This class defines the most basic attributes of a QReact code in a convenient object form and is used **internally** to store and process information about the QR codes defined and used by the system. Certain attributes are obligatory whereas others are optional. Table 3.51 presents a list of said attributes.

In addition to these, certain methods are implemented which copy, check and return information about the *QRObject* in question. Most of them are the expected `get/set` pair of methods to act upon the parameters just described.

Most methods are implemented using `String`s. Additionally, however, support for serialization had to be built into these objects. This was accomplished by using `HashMap`s from the standard `java.util` package where appropriate. To illustrate this point, two specific methods from *QRObject* are shown: `asHash()` and one of the class's constructors. There is another relevant point to these examples: as can be seen in listings 3.3 and 3.4, Java generics are heavily used to ensure type safety at compile time, thereby facilitating the overall debugging process.

name	*type*	*obligatory?*	*usage*
idTag	string	yes	unique identifier of the QR code, following the format defined in *QRDefs*
displayName	string	yes	descriptive name given to the code. Relevant to the user, **not** used by the system
programName	string	yes	activity associated with the QR code
helpTag	string	no	further information associated with the code, if desired
numParams	string	yes	number of parameters the QR code contains
complex	boolean	n/a	an **internal** parameter, states whether more than one parameter is defined in the QR code

Table 3.51.: *QRObject* attributes

```
public QRObject (HashMap<String, String> cnt) {
   idTag = cnt.get("id");
   displayName = cnt.get("name");
   programName = cnt.get("prog");
   helpTag = ((cnt.get("tag") == null) ? "" : cnt.get("tag"));
   numParams = cnt.get("params");
   complex = numParams.contains(",");
}
```

Listing 3.3: `QRObject` constructor using `HashMap`

```
public HashMap<String,String> asHash() {
   HashMap<String,String> res = new HashMap<String,String>();

   res.put("id", idTag); res.put("name", displayName);
   res.put("prog", programName); res.put("tag",helpTag);
   res.put("params", numParams);
```

```
    return res;
}
```

Listing 3.4: `QRObject` method `asHash()`

Needless to say, every one of the possible uses of *QReact* necessarily has to go through the *QRObject* class.

3.3.2. QRList

In addition to building upon the functionality offered by *QRObject, QRList* has the following objectives:

1. to coordinate the input/output aspects of all code definitions used in *QReact.* The actual I/O operations are performed by methods from other classes but *QRList* defines when these are to be executed. *QRConfig*, described in 3.3.4, is used for this purpose
2. to handle list-like operations such as addition, editing, removal and counting of QR definitions
3. to provide information relevant to the elements stored, such as what activities are required in the codes and how many of these can be executed
4. to provide format checks to ensure that only valid QR definitions are operated on, thereby preventing possible run-time format-related mistakes
5. to store the IDs of all usable codes
6. to store the IDs of all codes whose use has been attempted, regardless of whether they were passed on to the security system or ignored due to errors of any kind

It is important to note that while it is in principle possible to use this component on its own, it is intended to be used as a part of *QReact* and it might not be immediately clear what its purpose might be outside of the framework.

3.3.3. QRDefs and configuring the QR code formats

The main purpose of this class is to provide format definitions. They are the core of *QReact* and must therefore be tested thoroughly to ensure they function properly. This class also provides the following sets of methods:

- to validate the QR codes produced, in varying levels of depth
- decapsulation method, which strips the QR code into its payload components for posterior parsing
- a basic method (`QRDefs.getUniqueID()`)to provide the all-important unique identifiers of the QR codes
- a **very** basic method (`QRDefs.encrypt(String content)`) to provide an encryption layer
- to create correctly formatted QR codes, by default encrypted and with a unique identifier

Programatically, this class consists of static methods and constant definitions. Making the definitions constant was achieved by modifying the `String` literals with `public static final` identifiers, thereby ensuring they can only be modified in this particular file. The class itself was programmed to be `public` (but not `static`) to facilitate the testing process later on.

As detailed in section 3.1, one of the core requirements of *QReact* is complete freedom to define the syntax of the QR codes in use. This is accomplished precisely in this software component. Every aspect of the formats can be changed by adapting the definitions in this file to the scenario in question. Table 3.52 shows what can be configured in the *QRDefs* module, whereas listing 3.5 shows a possible configuration, one of the default ones used in the testing process.

(Re-)configuration issues pertaining to the formats

1. Special care should be taken to ensure that the changes make sense programatically. For instance, if an additional command is called for, it must not only be defined in its `String` form but also included in the `COMMANDS` array. Similarly, if a command's name is to be changed, the change needs to be reflected not only in the description but in the `String` array as well. In either case, these changes must be also programatically reflected in the *QReact* module, described in section 3.3.7, to make the commands usable.
2. *QReact* has very strict checks regarding the format of the QR codes (user side of the equation) but no "sanity checks", meaning that there is no (offered) way to check if the programmer is using valid commands.

3. Programmers are **strongly** advised to change the contents of the `QRDefs.encrypt()` method. The one provided by default is a simple variation of a letter-substitution cipher (ROT 25) that does enough to obfuscate the contents of the code but by no means provides true security.
4. To change the format of the identifiers, **both** their new format **and** the contents of the `QRDefs.getUniqueID()` method need to be updated coherently with each other, for the changes to take effect.

```
public static final String QRCODE_START_SQ = "!#RCT";
public static final String QRCODE_END_SQ = "!";
public static final String QRCODE_DELIM = "#!";
public static final String QRCODE_FMT =
   "^(" + QRCODE_START_SQ + ")([\\d]{4})([a-zA-Z0-9#,␣!?])*"
   + QRCODE_END_SQ+"$";
public static final int LEN_CMD = 4;
public static final String PARAM_FMT = "[A-Za-z0-9␣.]+";
public static final String PARAM_DELIM = ",";
public static final String ID_FMT_1 = "[a-z0-9]{6}";
public static final String EDIT_CMD = "0341";
public static final String ADD_CMD = "0981";
public static final String PASS_CMD = "7614";
public static final String REMOVE_CMD = "7647";
public static final String[] COMMANDS = { EDIT_CMD, ADD_CMD,
   PASS_CMD, REMOVE_CMD };
public static final String NO_ERROR = "0000";
public static final String ERROR_FOUND = "1000";
```

Listing 3.5: A possible configuration built into `QRDefs`

parameter	*usage*
`QRCODE_START_SQ`	Starting symbol or sequence of QR code
`QRCODE_END_SQ`	Ending symbol or sequence of QR code
`QRCODE_DELIM`	Sequence or symbol to delimit the different parameters
`QRCODE_FMT`	A regular expression, formatted according to the rest of the parameters parameters. Every QR code will be tested against this string
`LEN_CMD`	An `int` showing the number of commands allowed
`PARAM_FMT`	A regular expression that shows the parameter format
`PARAM_DELIM`	If there is more than one parameter, this is the delimiting string between them
`ID_FMT_1`	A regular expression which specifies the identifier format
`EDIT_CMD`	The edit command string
`ADD_CMD`	The add command string
`PASS_CMD`	The pass command string
`REMOVE_CMD`	The remove command string
`COMMANDS`	An array of strings containing the previous four parameters
`NO_ERROR`	A string specifying no error was found during the parsing and/or interpretation of the code
`ERROR`	A string specifying at least one error was found during the parsing and/or interpretation of the code

Table 3.52.: `QRDefs` configuration options

3.3.4. QRConfig

This component was left in the final version of the code for compatibility reasons and also to contribute to a more ordered component structure. Its function is to provide system-wide settings and to keep them separate from the QR code settings. More specifically, its features are as follows:

- configurable location of any configuration files deemed necessary. This includes target sensing which permits different locations depending on the target architecture and can be changed as needed.

- abstracted read and write operations. Although these are a different component's responsibility, *QRConfig* controls when they are to be executed.

The presence of *QRConfig*, although it might seems superfluous at first sight, facilitates *QReact*'s portability by having all OS-related settings in the same file, similarly to what *QRDefs* does with the formats. Therefore, if *QReact* is to be ported to another system or embedded into a different system, this is the file which holds the setting which will need to be changed accordingly.

3.3.5. YAML I/O modules

YAML (*Yet Another Markup Language,* `http://www.yaml.org`) is a very powerful alternative to XML frequently used for object serialization, configuration files and system-wide settings. In its original form, it supports associative arrays, nesting and multi-level lists. A very reduced subset of YAML is used in *QReact* to store information about the available and used QR codes. The two modules in question, *YAMLReader* and *YAMLWriter*, were written from scratch specifically for this project and provide a minimalist, although completely functional, ad-hoc implementation of the YAML format.

This module would probably need to be (partially) modified if *QReact* were to be ported to a different architecture. Embedding it into a different system would not call for any modification.

3.3.6. QRException

The shortest module in this framework, *QRException* provides a standard way to communicate with the *QReact* module and/or with all other modules that use it. It allows the calling module to receive informative messages as to any exceptions that occur during the parsing and execution process of a given QR code.

Listing 3.6 shows the contents of *QRException.*

```
public class QRException extends Exception {
   private static final long serialVersionUID = 1L;
   private String text;
   public QRException() {
      super();
   }
   public QRException(String message) {
        super(message);
        text = message;
    }
    public String getExceptionInfo() {
        return text;
    }
}
```

Listing 3.6: `QRException` source code

3.3.7. QReact

Finally, the framework provides a component, aptly named *QReact*, which encapsulates all functionality offered by the previous modules into an object that can be instantiated and used to parse and act upon any QR code presented to it. *QReact* is responsible for the following:

1. rigorous syntax checks, for which it uses *QRDefs.*
2. parsing of the various commands, backed up by rigorous semantic checks. These are performed individually depending on the command which needs to be dealt with and are hard-coded into this component.
3. if needed, to update the list of codes (using *QRList*) and the pertinent files (using *QRConfig*)
4. to provide a pre-parse semantic and syntax checks on demand (method `isQROk()` shown in listing 3.7). This method shows possible mistakes to the user without raising any exceptions.
5. to handle the pertinent exceptions as the arise.
6. in general, to act upon the code's contents.

As an example of how some of this functionality is implemented, listing 3.7 shows how *QReact* interacts with the rest of the framework's components. This function returns a `HashMap` object which contains information about the correctness of the given QR code. If there are any errors, it sends informative messages to the caller.

It is worth mentioning that this function detects these errors *before* the code is acted upon. The rest of the methods return specific information in the form of a *QRException,* both before the code is acted upon and during “execution”. This would roughly be the equivalent of a compile-time and run-time errors.

```
public HashMap<String,String> isQROk(String s) {
   HashMap<String,String> res = new HashMap<String, String>();
   res.put("result",QRDefs.NO_ERROR); // no errors by default
   tokens = s.split(QRDefs.QRCODE_DELIM);
   if (tokens.length < 4 ) {
      res.put("result",QRDefs.ERROR_FOUND);
      res.put("message", "Not␣enough␣parameters\n");
   }
   if( QRDefs.isValidQRString(tokens) &&
         QRDefs.isValidCommand(tokens) ) {
      switch (tokens[1]) {
         case QRDefs.EDIT_CMD :
            res.put("code", QRDefs.EDIT_CMD);
            res.put("oper", "edit");
         break;
         case QRDefs.ADD_CMD  :
            res.put("code", QRDefs.ADD_CMD);
            res.put("oper", "add");
         break;
         case QRDefs.EXEC_CMD :
            res.put("code", QRDefs.EXEC_CMD);
            res.put("oper", "exec");
         break;
         case QRDefs.REMOVE_CMD :
            res.put("code", QRDefs.REMOVE_CMD);
            res.put("oper", "remove");
         break;
      }
      res.put("ID",tokens[2]);
   }
   else {
      res.put("result",QRDefs.ERROR_FOUND);
      res.put("message", "Invalid␣syntax␣or␣command!!\n");
   }
```

```
    return res;
}
```

Listing 3.7: Method `isQROk()` from `QReact`

3.3.8. Adapting the formats and embedding *QReact*

As a summary of this section, table 3.53 shows whether a part of the individual modules will need to be adapted, significantly or otherwise, if a change in formats or embedding is necessary.

component	*format change*	*embedding*	*porting*
QRObject	probably	no	no
QRList	no	no	no
QRDefs	yes	no	no
QRConfig	no	yes	yes
YAML I/O modules	no	no	probably
QRException	no	no	no
QReact	probably	no	no
GUI	probably	probably	yes

Table 3.53.: Suggested ways to adapt *QReact*

In the light of this information, it can be concluded that this framework can be adapted to new uses and scenarios fairly easily. No significant changes need to be made outside of certain modules, except the possible GUI[2], which by its very nature is platform and scenario-dependent in every case.

3.4. Tests

Once the system has been implemented, it is important to verify that it functions properly. Given the strong modular design of *QReact*, every one of the modules was tested individually as it was developed. Next, a batch of tests was executed to verify the correctness of the overall system, utilizing *QReact*'s built-in safety checks, as well as online format-checking services. Stress tests were also executed, whose number was limited due

[2]Strictly speaking, the GUI is not a part of the framework, it is just included here for completeness

to the nature of the project. In order for those to be more meaningful, they should be executed on the complete system, once *QReact* is embedded into it.

3.4.1. Functional tests

Functional tests are executed in order to verify that the implementation complies with the design characteristics. In this project, special attention was paid to the formats and the built-in safety features of the framework in terms of exception handling and improper formation of the QR codes since it was deemed that these are the crucial aspects of an access control system.

In order to facilitate the reading of the results and their link to the design requirements, the tests are presented in tables which all have fields with the following format:

Name is a descriptive name of the test

ID is the unique identifier of the test. For these tests, its format is **FTXX**, where **XX** is the number of the test

System identifies the module which was tested. At times, there was more than one module being tested

Result can be either *pass*, *satisfactory* or *fail*

Description provides a brief description of the test

Steps provides a detailed description of all steps taken in this test

For the sake of brevity, only the most relevant tests and their results are described in the upcoming pages.

name	**Functional test 1**	*ID*	**FT01**
system	**QReact**	*result*	**pass**
description	1. Overall test with 180 QR codes, 60% of which correctly formed 2. Additionally, test with 50 QR codes which were not used before		
steps	1. Formulate random contents for 180 codes 2. Run `QRDefs.makeCode()` to produce 108 correct codes 3. Randomly produce 72 additional QR codes 4. Get a random number from 1 to 50 and decide if it will signify the correct or incorrect items, then produce the appropriate number of items 5. Present QR codes to scanner, one at a time, and record results		

Table 3.54.: Functional test 1

name	**Functional test 2**	*ID*	**FT02**
system	**QRObject**	*result*	**pass**
description	Complete core test of the `QRObject` module and its functionality		
steps	1. Formulate random contents for 100 codes 2. Create 100 different `QRObject`s 3. Test every one of the methods, content-managing and information, of this module and record results		

Table 3.55.: Functional test 2

name	**Functional test 3**	*ID*	**FT03**
system	**QRList**	*result*	**pass**
description	Complete core test of the functionality of `QRList` using the `QRCode` objects from **FT02**		
steps	1. Create a `QRList` object, populate with the 100 `QRObject` objects produced in **FT02** 2. Test all functions of the `QRList` module and record results		

Table 3.56.: Functional test 3

name	**Functional test 4**	*ID*	**FT04**
system	**QRDefs**	*result*	**pass**
description	Internal test to verify `QRDefs` code definitions against format changes		
steps	1. Change the regular expressions containing the format of the QR codes, including identifier length 2. Validate the new format using `QRDefs.validate()` 3. Repeat steps 1-3 five times, each time with a new format		

Table 3.57.: Functional test 4

name	**Functional test 5**	*ID*	**FT05**
system	**QRDefs**	*result*	**pass**
description	Internal test for equivalence of definitions when validated using regular expressions (default) and regular string manipulation functions		
steps	Two versions of the validation functions were produced: one using regular expressions and another using string manipulation functions. Test for equivalence using the definitions in **FT04.** The default chosen was regular expressions		

Table 3.58.: Functional test 5

name	**Functional test 6**	*ID*	**FT06**
system	**YAMLReader**	*result*	**pass**
description	Internal test for proper reading of the reduced YAML format used to exchange and export `QRObjects`		
steps	1. Create 80 random `QRObjects`, 60% of which correct, and save them in a YAML file 2. Read all definitions in said file and record exceptions as results		

Table 3.59.: Functional test 6

name	**Functional test 7**	*ID*	**FT07**
system	**YAMLWriter**	*result*	**pass**
description	Externally validated internal test of proper formatting of the reduced YAML format used to exchange and export `QRObjects`		
steps	1. Recover file produced in **FT06** 2. Run every one of the definitions and the whole file through the YAML validator at `www.yamllint.com` and record exceptions as results		

Table 3.60.: Functional test 7

name	**Functional test 8**	*ID*	**FT08**
system	**QRConfig, QReact, QRList**	*result*	**pass**
description	Overall test of reading and writing files in different scenarios		
steps	1. Recover file produced in **FT03**, place it in a network storage, SD card, standard location and online 2. Test for proper function in every one of the above mentioned scenarios		

Table 3.61.: Functional test 8

name	**Functional test 9**	*ID*	**FT09**
system	**QRDefs**	*result*	**pass**
description	Internal test of the creation feature of `QRDefs,` which impacts the system's overall performance		
steps	1. Create 60 QR codes using `QRDefs.makeCode()` 2. Validate said codes using `QRDefs`' functions and record results		

Table 3.62.: Functional test 9

name	**Functional test 10**	*ID*	**FT10**
system	**QReact, QRList, YAMLWriter, YAMLReader**	*result*	**pass**
description	Verify that the export feature of QReact produces valid YAML files		
steps	1. Using the codes produced in **FT09**, call `QRList`'s `getUsableCodes()` 2. Validate said results using the online validator at `www.yamllint.com` and record results		

Table 3.63.: Functional test 10

name	**Functional test 11**	*ID*	**FT11**
system	**QReact, QRList, YAMLWriter, YAMLReader**	*result*	**pass**
description	Verify that the used codes feature of QReact produces valid YAML files		
steps	1. Using the codes produced in **FT09**, call `QRLists`'s `getUsedCodes()` 2. Validate said results using the online validator at `www.yamllint.com` and record results		

Table 3.64.: Functional test 11

name	**Functional test 12**	*ID*	**FT12**
system	**QReact, Android, ZXing**	*result*	**pass**
description	Verify that QR codes can be read from low-resolution sources and in poor ambient conditions		
steps	Show the codes produced in **FT01** on a screen with low resolution and in low light to test the camera's ability to focus and ZXing's ability to interpret the codes properly		

Table 3.65.: Functional test 12

name	**Functional test 13**	*ID*	**FT13**
system	**QReact, QRException**	*result*	**pass**
description	Verify that malformed QR codes are properly identified		
steps	Show the malformed codes produced in **FT01** to make sure the right `QRException` is thrown every time and record results		

Table 3.66.: Functional test 13

name	**Functional test 14**	*ID*	**FT14**
system	**QReact, AdminConsole**	*result*	**pass**
description	Overall test of `AdminConsole` to verify correct function		
steps	Test overall usability and GUI responsiveness by using AdminConsole in a simulated environment		

Table 3.67.: Functional test 14

3.4.2. Traceability matrix of functionality

Table 3.68 shows the correspondence between the software components and tests performed. Provided that there is at least one tick per component, its functionality is correct per the tests performed. A few tests of overall functionality were performed which guarantees that all components have been tested and were found to function properly.

	FT01	**FT02**	**FT03**	**FT04**	**FT05**	**FT06**	**FT07**	**FT08**	**FT09**	**FT10**	**FT11**	**FT12**	**FT13**	**FT14**
SRNF01	✓			✓	✓			✓	✓	✓	✓	✓	✓	✓
SRNF02	✓			✓	✓			✓	✓	✓	✓	✓	✓	✓
SRNF03	✓					✓	✓	✓		✓	✓	✓	✓	✓
SRNF04	✓			✓	✓			✓	✓	✓	✓	✓	✓	✓
SRNF05	✓	✓	✓					✓		✓	✓	✓	✓	✓
SRF06	✓	✓	✓			✓	✓	✓		✓	✓	✓	✓	✓
SRF07	✓	✓	✓			✓	✓	✓		✓	✓	✓	✓	✓
SRF08	✓					✓	✓	✓		✓	✓	✓	✓	✓
SRF09	✓			✓	✓			✓	✓	✓	✓	✓	✓	✓
SRNF10	✓							✓		✓	✓	✓	✓	✓
SRF11	✓							✓		✓	✓	✓	✓	✓
SRNF12	✓			✓	✓			✓	✓	✓	✓	✓	✓	✓
SRF13	✓			✓	✓			✓	✓	✓	✓	✓	✓	✓
SRF14	✓			✓	✓			✓	✓	✓	✓	✓	✓	✓
SRNF15	✓			✓	✓			✓	✓	✓	✓	✓	✓	✓
SRF16	✓			✓	✓			✓	✓	✓	✓	✓	✓	✓
SRF17	✓			✓	✓			✓	✓	✓	✓	✓	✓	✓
SRF18	✓			✓	✓			✓	✓	✓	✓	✓	✓	✓

Table 3.68.: Traceability matrix of functionality

4. Planning and budget

"Never confuse the size of your paycheck
with the size of your talent"
Marlon Brando

This chapter describes in detail the planning followed when developing this project. A Gantt diagram is presented to give a better vision of the time employed in the planning, development and debugging phases. Special attention is paid to the software used, most of which is free and open source. A detailed account of the expense, both hardware and human, is presented and the overall cost of the project is thereby calculated.

4.1. Planning

The starting date for this project was August 18th 2014. Table 4.1 lists the days, in addition to the weekends, that have been considered non-working. A detailed report of the time employed in the development of *QReact* is given in table 4.2. Based on this data, figure 4.1 represents the Gantt diagram associated with *QReact*. Notice that owing to the specificity of the software used for the project management, the diagram also represents whom every task was assigned to.

period	*days*	*explanation*
8/9/14	1	local festivity
8/12/14	1	local festivity
24/12/14 - 6/1/15	10	Christmas break
30/3/15 - 3/4/15	6	Easter break
6/4/14	1	local festivity

Table 4.1.: Non-working days during the project development

task	*started*	*finished*	*days*
Feasibility study	18/8/14	29/8/14	10
Analysis of requirements	1/9/14	5/9/14	5
Specifications (preliminary documentation)	9/9/14	10/9/14	2
Architectural design	11/9/14	16/9/14	4
Interface design	17/9/14	22/9/14	4
Component design	23/9/14	9/10/14	13
File formats	10/10/14	13/10/14	2
QR code formats	14/10/14	15/10/14	2
Implementation and testing	16/10/14	23/3/15	102
Implementation of example program	24/3/15	14/5/15	10
Entire system testing	15/4/15	18/5/15	24
Documentation	19/5/14	13/7/15	40

Table 4.2.: Tasks involved in the project

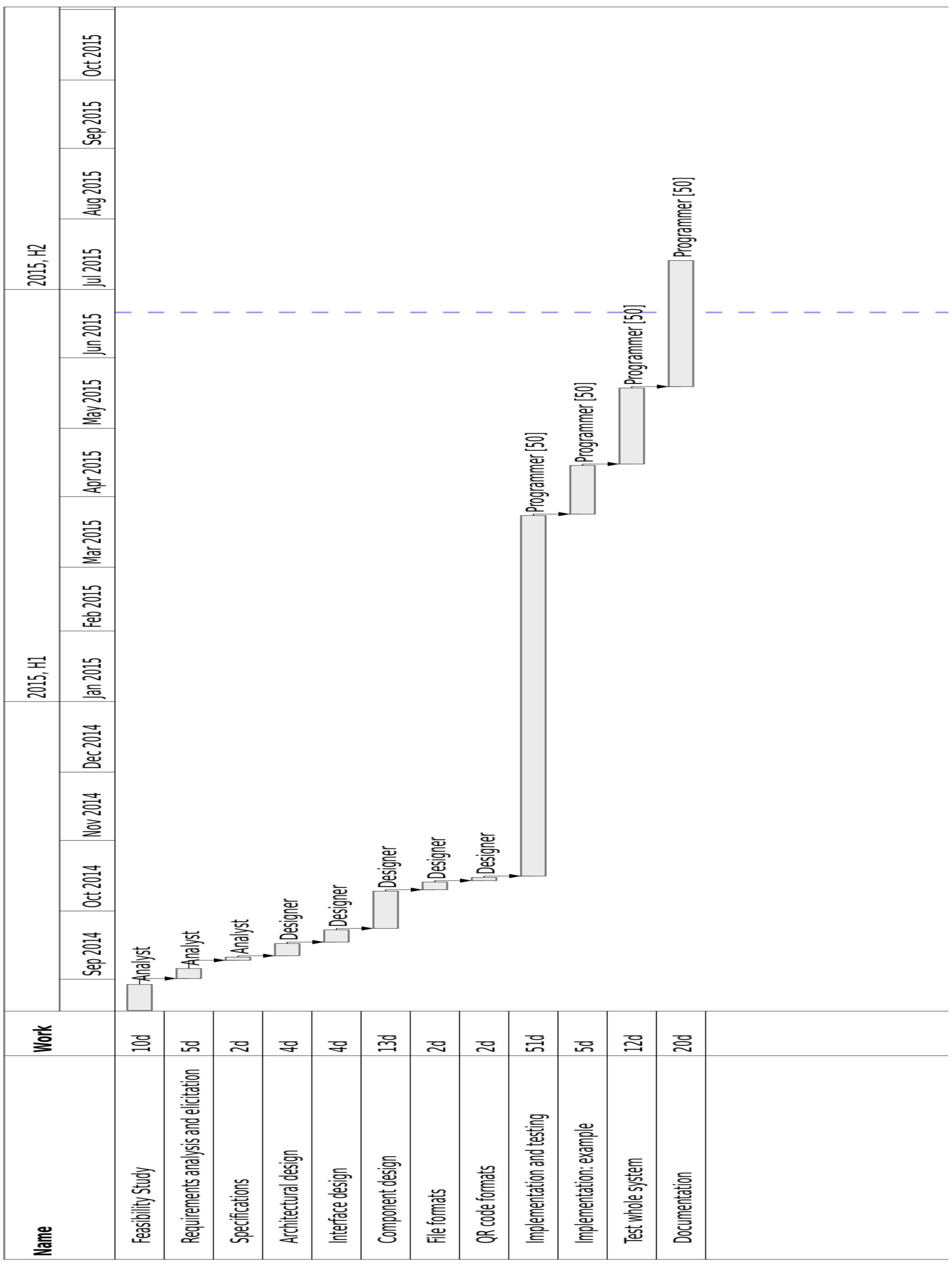


Figure 4.1.: Gantt of the project

4.2. Personnel cost

The personnel cost involved in developing this project comes from careful consideration of the number of hours and the type of position involved.

The first step is to determine the personnel involved in the development of this project and their hourly salary. Using the information found in [12], table 4.3 can be compiled. It is important to note the dedication: full time implies 8 hours a day whereas part time means 4 hours a day.

name	*position*	*rate*	*dedication*
Atanas Plamenov Karaguiozov	system analyst	15.50 €/h	full time
Atanas Plamenov Karaguiozov	software designer	13.26 €/h	full time
Atanas Plamenov Karaguiozov	programmer	12.63 €/h	part time

Table 4.3.: Personnel and hourly rates

Next, the total number of hours dedicated to the project is computed, as well as their cost. Table 4.4 contains the relevant calculations. Note that it is essentially an augmented version of table 4.2. To convert from working days to working hours, the dedication parameter from table 4.3 has been used.

task	*days*	*hours*	*rate (€/h)*	*cost (€)*
Feasibility study	10	80	15.50	1240
Analysis of requirements	5	40	15.50	620
Specifications (preliminary documentation)	2	16	15.50	202.08
Architectural design	4	32	13.26	424.32
Interface design	4	32	13.26	424.32
Component design	13	104	13.26	1379.04
File formats	2	16	13.26	212.16
QR code formats	2	16	13.26	212.16
Implementation and testing	102	408	12.63	5153.04
Implementation of example program	10	40	12.63	505.20
Entire system testing	24	96	12.63	1212.48
Documentation	40	160	12.63	2020.80
			Total	13605.60

Table 4.4.: Detailed personnel costs excluding *VAT*

The personnel cost, therefore, comes up to $13605,60$ **€**.

4.3. Hardware cost

In order to develop this project, a certain amount if hardware had to be purchased. To test the system, both new and existing hardware was used.

An important parameter to take into account with regards to hardware costs is depreciation. One way to view depreciation is as the amount of time an organisation expects to run a piece of equipment until it is considered obsolete. For the following calculations, we define the depreciation period to be 60 months, or five years. Given today's rhythm of innovations and the constantly lower prices of electronic equipment, five years is definitely a conservative estimate for depreciation. It was chosen, however, to reflect the fact that for all phases of this project, computational power is not of utmost importance. The hardware was also chosen to be modern enough so it is expected to be run for longer before it becomes obsolete.

The following formula is used to calculate the hardware cost per item:

$$\pi_i = \frac{\alpha}{\delta}\epsilon\rho \tag{4.1}$$

where

α is the number of months since the item was purchased that it was in use,

δ is the depreciation period, 60 months in this case,

ϵ is the item's initial cost in Euros, and

ρ is the item's usage in the project, expressed as a percentage.

To calculate the total hardware cost, one just sums the individual contributions:

$$\pi = \sum_i \pi_i \tag{4.2}$$

Table 4.5 contains a list of all hardware involved and the associated per-item amortization costs, calculated using formulas 4.1 and 4.2.

item	*price*	*use (%)*	*months used*	*per-item cost*
Laptop *Acer E15*	378	100	11	69.30
Monitor *Samsung T24B300*	181.48	100	11	33.27
Tablet *Samsung Galaxy S*	298.62	100	5	24.89
Mobile phone *Samsung Galaxy S4*	260.70	100	5	21.73
			Total	149.18

Table 4.5.: Hardware cost excluding *VAT*

It is important to note that the tablet and the mobile phone were used to program and test the project under two different versions of Android only after the framework was developed. Therefore, they were used for the second phase of the project compared to the other two items, which were used throughout the project development.

The total hardware cost is therefore 149.18 €.

4.4. Software cost

One of the peculiarities of this project is that it uses exclusively open source software. The development platform, the document preparation system, the debugging platform and the target devices all use software whose economical impact on this project is non-existent. No purchases were necessary as everything was readily available online or from the operating system's repositories.

Table 4.6 lists the software used in the various stages of the project. One can easily conclude that the software cost is 0 €.

item	*licence*	*cost*
Operating system *LinuxMint 17*	open source	0
Eclipse IDE 3.8	open source	0
Android Studio 1.2.2	open source	0
Planner 0.4.16	open source	0
Task Coach 1.3.36	open source	0
LyX*2.0.6*	open source	0

Table 4.6.: Software cost

4.5. Total project cost

To obtain the total cost of this project, the personnel, hardware and software costs must be added. A 20% in indirect costs to cover unexpected expenses, as well as those not considered in the budgeting process, is next added. Finally, *VAT* is applied.

The result is presented in table 4.7.

description of cost	*value*
Personnel	13605.60
Hardware	149.18
Software	0
Indirect cost (20%)	2750.96
VAT (21%)	3466.20
Total	19971.94

Table 4.7.: Total project cost

The total cost of this project is therefore 19971.94€.

5. Conclusions and future development

"Just because you got the monkey off your back doesn't mean the circus has left town"
George Carlin

This chapter present the conclusions reached after the development phase was completed. The degree to which the objectives have been completed is examined. A specific mention is given to the personal conclusions. The chapter concludes with a number of possible suggestions for future development of *QReact*.

5.1. Objective conclusions

As already mentioned, setting objectives for a framework is different from setting objectives for other types of software. Therefore, the analysis and design stages had to establish the objectives even more clearly and unequivocally. This was a difficult task given that the focus had to be on functionality but it also involved thinking about what functionality had to be included, which in turn involves thinking about possible uses, then functionality again... Despite this seemingly infinite loop which might trap one during the core stages of the project however, the resulting framework has all the bits and pieces that are necessary for a more than satisfactory function.

- The core part of this project was **modularity**. *QReact* was designed as a modular structure in which all elements interface tightly. Moreover, if any one of the modules is taken away from the structure, it is not immediately usable, which proves the tight interface between them.

- Another key point was the **ease of configuration**. To accomplish this, a class consisting of static methods was created, which is used by every one of the other classes. This means that if any changes need to be made, only one file needs to be altered accordingly and the changes will "propagate" across the framework.
- The previous two objectives facilitate the **high level of customizability**. Anything can be changed, from the location of the configuration files to even a complete redefinition of the formats for the QR codes.
- The modular design helped to accomplish another objective: the **ease of portability**. Only those parts of the framework which are platform-specific need to be re-written, the rest can be ported as it is.
- As another consequence of its modular design, *QReact* offers different **layers of functionality**; in this case, a number of layers of encryption are possible.
- As detailed in section 4.4, **only open-source software** was used. This was an important objective as it is in line with the developer's philosophy and with the platform used to deploy this framework on.

These were the objectives set at the beginning of the project but they are by far not the ones to be completed. Due to the abstract nature of *QReact,* as the various parts were developed, more and more ideas came to light which facilitated and improved on the initial design in terms of structure, robustness and possible uses. The project planning was followed as strictly as possible and the framework was finished in the time allotted. It is also important to point out that the budget was more than enough and there were no overruns.

As a result, it can be clearly and unequivocally stated that all objectives were more than satisfactorily achieved.

5.2. Personal conclusions

My idea about this project was to take it up as a personal challenge and an opportunity to learn as much as possible. I must convincingly say that this happened beyond my most ambitious plans. Any software project is more than programming and now I know it from personal experience.

As regards the programming part, Android has always been a fascinating world for me. No previous knowledge about mobile operating systems was an advantage to a certain extent, as everything I was reading seemed to unveil a different organisation. My initial interest was more into the GUI aspect but this quickly changed. The complexities of Android quickly became a fascinating insight into component interoperability. I discovered that my own way of thinking, both personal and professional, was ideally suited to using such a well organized system from within. As a Linux and open-source enthusiast, Android for me was the clear choice from the very beginning. The "scratch-your-personal-itch" philosophy of open source software and the availability of so much code to look at and learn from definitely helped but it was the unique combination of curiosity, discovery and novelty that propelled these early days of the project. As things started taking shape and unexpected problems had to be tackled, it seemed like the old saying of "Every program tends to grow in complexity until it exceeds its author's capabilities" was getting dangerously close to turning out to be true. But then, it is the idea that you have created something and that you need to give it a well-defined shape, and your best shot that keeps you going and learning more and more. Overall, the experience was great.

In addition to programming, I got the opportunity to use some programs for a lengthy and extensive purpose. LyX and LaTeX were definitely not new to me but I had never used them to write such a long document as this report. Using Planner was a unique experience: I learned it in one afternoon and I still can't believe that a program so simple has such great functionality and can save me the trouble of cost calculation and the Gantt chart. Finally, the transition from Eclipse to Android Studio, necessitated by the project and recommended by Google, was full of headaches and frustration at first but I can now safely add it to the list of programs I am comfortable using. It added to the complexity of this project and the very positive outlook on trying new things.

To conclude this section, it is undoubtedly fascinating that a project can help you accomplish so many things at once.

5.3. Future lines of development

The author hopes that this is not the end for *QReact*. A framework should definitely continue to evolve and be updated with the requirements of today's world. Many facilities to port and configure this framework have been included in the design and in principle, it would be only a matter of changing a few definitions in the *QRDefs* file to customise the system to the applications it is needed for.

Following is a list of possible lines of development.

- To fully illustrate *QReact*'s potential for portability and interoperability, it can be ported under various operating systems and hardware platforms different from Android:
  - implementing under Android for Raspberry Pi would be readily accomplished. It can be used in conjunction with the camera module sold separately for rapid prototyping of any kind of security system based on *QReact*
  - mentioning Raspberry Pi, one can't help but think about the plethora of embedded systems that can be readily found on the market today. There are different Java compilers, even a number of "Java processors" that can run on embedded systems[30][1]. The requirement is for the *QReact* to be re-compiled and the relevant input/output functionality adapted to suit the platform in question.
  - of course, one shouldn't forget the *iOS* ecosystem. Even though this platform was not chosen at the beginning of the process, it is an important player on the market and possible applications can surely be built on top of it. Following is a list of products which have[2] a free-to-use option for open-source projects and can be used to accomplish the task:

    Codename One (`http://www.codenameone.com`) is a well established compiler that can target various mobile platforms including *iOS*.

    RoboVm (`http://robovm.com/`) is another compiler and IDE which supports full hardware access and ample pre-defined fine-tuning capabilities.

    Avian (`http://oss.readytalk.com/avian/`) goes even further: it is a whole lightweight virtual machine and class library which can be easily ported for various platforms.

    XMLVM (`http://www.xmlvm.org/overview/`) is an extensible cross-compiler tool-chain, where all bytecode is stored as an XML file. It can cross-compile Android applications for *iOS*.
  - *Windows Mobile* is another possible target that should be mentioned, even though it doesn't fully conform to the Linux/open source idea of the project. Porting *QReact* on it, however, should prove quite easy once the GUI and input/output components are handled properly.

[1] Even though this reference is somewhat outdated, it provides a useful starting point on the subject
[2] As of June 2015

- In terms of expandability, a lot can be achieved by changing the definitions of the QR codes in use. Some examples follow:
  - in the field of access control, given that there are a number of set-top boxes and TV sets running Android firmware, those of them which have a camera can use it to control access to the TV or the contents depending on the time of the day, the QR code presented, etc.
  - various systems for voucher-like transactions or money transfers can be developed. The codes can be created at one end of the system and redeemed or otherwise used at the other.
  - systems relying on scanning a unique code with a time-stamp can be easily implemented. For example, tourist organisations tend to have programs[3] which need evidence that certain places of interest have been visited. *QReact* can be readily used to facilitate said proof.
- To further enhance features, one possible and relatively easy enhancement to *QReact* is the possibility of incorporating different encryption algorithms, possibly even more than one, to be used for different purposes. This is relatively easy as the "encryption layer" is embedded into the *QRDefs* file and the change will be just a function call away. In addition, and to make things a bit easier, Java and Android both provide a native way to execute *C* code so if necessary for this purpose, JNI can be used as described in [8].
- As another feature enhancement, ZXIng's built-in capabilities for "scanning" the QR codes from an HTML source can be taken advantage of. Functionality could be added so that the source of the symbols is not just the camera but emails, XML files or maybe even Base64-encoded QR codes.

Integration into a system for access control implies designing the system accordingly. A number of possible components will have to be added, which would have to do with where the credentials are stored, the way the resulting QR codes are shared and the access control proper. A distributed solution will certainly be called for.

QReact is offered with a certain amount of functionality. Adding to it is perhaps the most cumbersome part of extending the framework. Naturally, the complexity of such an endeavour will depend upon what needs adding but modifications to at least the *QReact* class will most probably have to be involved.

[3]For one such program, see [20]

Last but definitely not least, it is hoped that *QReact* will be expanded upon and embedded into many projects over its lifetime.

A. Android Framework

This appendix briefly describes what Android is. A historical account is given, followed by a brief description of some of the internal components.

A.1. What is Android

A.1.1. Brief history

Android OS dates back to October 2003, when *Android, Inc.* was founded in Palo Alto, California by Andy Rubin, Rich Miner, Nick Sears and Chris White. The idea, in Rubin's words, was for them to develop "...smarter mobile devices that are more aware of its owner's location and preferences." All founders had previous experience and important accomplishments behind their backs but the new company remained secretive when it came to what they were doing, revealing only that it was working on software for mobile phones.

In August 2005, *Android Inc.* was acquired by Google, making it a wholly owned subsidiary of *Google Inc.* This move was interpreted to mean that Google was planning to enter the mobile phone market. As a part of *Google Inc.*, the team led by Rubin decided to develop a mobile device platform powered by the Linux kernel. The new platform was marketed and advertised to handset makers and carriers as one that would provide a flexible, upgradable system. At the same time, a number of hardware component and software partners were recruited to work with the project. Up until then, Google's intention to enter the mobile communications market was no more than speculation. It continued to build through December 2006.Various reports noted that Google wanted its search and applications on mobile phones and it was working hard to deliver that. Rumours soon followed that Google was developing a Google-branded handset.

On November 5, 2007, the Open Handset Alliance, a consortium of several companies among which *Broadcom Corporation, Google, HTC, Intel, LG, Marvell Technology Group, Motorola, Nvidia, Qualcomm, Samsung Electronics, Sprint Nextel, T-Mobile* and *Texas Instruments* was unveiled, listing as its goal the development of open standards for mobile devices. On the same day, their first product, Android, was unveiled.

On December 9, 2008, 14 new members joined, including *ARM Holdings, Atheros Communications, Asustek Computer Inc, Garmin Ltd, Huawei Technologies, PacketVideo, Softbank, Sony Ericsson, Toshiba Corp,* and *Vodafone Group Plc.*

A.1.2. Android versions features

Version 1.0

Android 1.0 was released with the first Android device, HTC Dream on G1, on 23 September 2008. It had the following features:

- Android Market application used to download and updates through the Market app
- Web browser with show, zoom and pan support for full HTML and XHTML. Multiple pages were shown as windows
- Camera support. Sadly, there was no way to change resolution, white balance, quality, etc.
- Folders, allowing the grouping of a number of app icons into a single folder icon on the Home screen
- Email with access to email servers commonly found on the Internet with support for POP3, IMAP4, and SMTP
- Synchronization with Gmail through the Gmail app
- Synchronization with Google Contacts through the People app
- Synchronization with Google Calendar through the Calendar app

- Google Maps with Latitude and Street View to access maps and satellite imagery and local business and driving directions through GPS
- Google Sync to manage over-the-air synchronization of Gmail, People, and Calendar
- Google Search of the internet and phone apps, contacts, calendar, etc.
- Google Talk instant messenger
- Instant messaging, text messaging, and MMS
- Media Player, which supported managing, importing, and playing back but had no video and stereo Bluetooth support
- Notifications appeared in the Status bar. It had to be drag down to see details, ringtone, LEDs and vibration options
- Voice Dialer allowed dialling and placing of phone calls without typing a name or number
- Wallpaper, allowing the user to set the background image or photo behind the Home screen icons and widgets.
- YouTube video player
- Plus other apps like Alarm Clock, Calculator, Dialer (Phone), Home screen (launcher), Pictures (Gallery), and Settings. This device supported WiFi, and Bluetooth.

Version 1.1

Android 1.1 was released on 9 February 2009 as an update on T-Mobile only. Aside from API changes and solved bugs, it included the following changes:

- Maps added details and reviews when a user does a search and clicks on a business
- Dialer: In-call screen time out default was longer when using the speaker-phone, incorporated was a Show/Hide Dialpad function
- Messaging supported saving attachments
- System supported for marquee in layouts.

Version 1.5 Cupcake

This version was based on Linux kernel 2.6.27 and was released as an update on 30 April 2009. Several new features and UI updates were included, such as the virtual keyboard, which now supported 3rd party keyboards with text prediction and user dictionary for custom words; video recording with the camera, video (MPEG-4 and 3GP) playback; stereo support on bluetooth and auto-pairing; copy and paste features in the browsers. Contacts also showed user picture for Favourites, events in the call-log were date and time stamped and a one-touch access to a contact card from call log event was introduced. Users could also have animated screen transitions and upload videos to YouTube and photos on Picasa. Another important innovation included in this version were the *Widgets*, miniature application views that can be embedded in other applications and receive periodic updates. Since then, the widgets have become an integral part of the Android user experience.

Version 1.6 Donut

On 15 September 2009, the 1.6 (Donut) SDK was released. It was based on Linux kernel 2.6.29. The most important updates were in the search capabilities, where text entry was complemented by the voice search and extended to cover bookmarks, history, contacts, the web, and more. Also, developers were allowed to include their content in search results. Conversely, advances were also made in the text to speech engine, allowing multi-lingual speech synthesis using different engines. This effectively allowed Android to “speak” a string of text. Search capabilities were also improved in the Android Market by allowing for easier searching, app screen shots, etc.

Version 2.x Eclair

The new family of versions was based on Linux kernel 2.6.29. The first of them, 2.0, was released on 26 October 2009. Major changes included the addition of multiple accounts for contact and email synchronization, Exchange support, flash support, digital zoom, colour effect and macro focus for the camera and HTML5 support for the browser. Improvements were made to the keyboard, a smarter dictionary was added which learns from word usage and includes contact names as suggestions. Hardware speed was optimized and support for more screen sizes and resolutions was added. This version features the inclusion of *Live Wallpapers*, which are animated home screen background images.

Two other minor versions exist, 2.0.1, released on 3 December 2009 and 2.1, released on 12 January 2010. They mostly featured minor bug fixes and were offered deployable to Android-powered handsets.

Version 2.2 Froyo

Froyo's latest release is 2.2.2, based on Linux kernel 2.6.32 and released on 20 May 2010. It featured various memory, performance and application speed improvements using JIT compilation. Chrome's JavaScript engine was integrated into the Browser application, as well as Adobe Flash support. Curious additional functionality introduced was USB tethering and Wi-Fi hotspot functionality and the option to disable data access over mobile network.

Version 2.3.x Gingerbread

Version 2.3 was introduced with the Google Nexus S terminal. It was based on Linux kernel 2.6.35 and released on 6 December 2010. It featured a UI speed-up, support for extra-large screen sizes and resolutions (WXGA and above), native support for VoIP telephony, new audio effects for the media player and further developments for audio, graphical, and input enhancements for game developers.

2.3.3 added several improvements and APIs to the Android 2.3 platform. 2.3.4 added support for voice and video chat using Google Talk. 2.3.5 brought improved network performance for the Nexus S 4G, among other fixes and improvements.

Version 3.x Honeycomb

Android 3.0 was based on Linux kernel 2.6.36 and was released on 22 February 2011. It is different from the rest in that it is a tablet-only release of Android. The first device working under Honeycomb, was the *Motorola Xoom* tablet, released on February 24, 2011. Its new features had to deal with the tablets and included optimized tablet support with a new virtual and "holographic" user interface, a new system bar at the bottom of the screen, improved multitasking, redesigned keyboard, tabbed browser windows, form auto-fill, and a new "incognito" mode. Other important features are the support for video chat using Google Talk hardware acceleration and support for multi-core processors

The next Honeycomb version, 3.1, was released on 10 May 2011. Aside from the UI refinements, it also featured resizeable home screen widgets and support for external keyboards and pointing devices, joysticks and game pads.

Version 3.2 was released on 15 July 2011 and featured improved hardware support, optimisation for a wider range of tablets, easier access for apps to files on the SD card and a compatibility display mode for apps that have not yet been optimized for tablet screen resolutions. It was also the version used by most first- and second-generation Google TV-enabled devices.4.x

Version 4.x Ice Cream Sandwich & Jelly Bean

The Linux kernel used for this version was 3.0.1 and it was released on October 19, 2011. Aside from refinements to the UI, it added WiFi Direct functionality, ubiquitous VPN, NFC features via AndroidBeam, improvements in the real-time dictation software and many more.

Versions 4.0.3 and 4.0.4, released on December 16, 2011 and Mar29 29th, 2012, respectively, contained mostly bug fixes, improvements and optimizations. Important fixes to the camera were also offered.

Ice Cream Sandwich was superseded by an incremental update called Jelly Bean, on July 9, 2012. Based on Linux 3.0.31, it mostly features improvements in the functionality and performance of the UI, in addition to support for USB audio, gapless playback and multichannel audio streams.

Jelly Bean 4.2, based on Linux 3.4.0, was announced via a press release, as a "new flavour of Jelly Bean" and released on November 13 2012. Aside from the compulsory UI improvements, bug fixes and tweaks, it added a complete re-write of the Bluetooth and NFC stacks and the possibility, in tablets only, of multiple users. USB debug white list and support for new Bluetooth devices was also added in the subsequent releases 4.2.1 (November 27, 2012) and 4.2.2 (February 11, 2013).

An "even sweeter Jelly Bean" 4.3 and its bug-fix version 4.3.1, were released on July 24, 2013 and October 3, 2013. They added Bluetooth low energy support, audio/video remote control profile and OpenGL E 3.0, which improved game graphics. Native support for the 4k resolution (up to 4096×2160 pixels) was supported for the first time. Various security improvements were also introduced.

Version 4.x KitKat

First released on October 31st, 2013, the next tasty treat of Android, 4.4, had a total of 8 different flavours (4.4.1, 4.4.2, 4.4.3, 4.4.4, 4.4W, 4.4W.1 and 4.4W.2), the last of which was introduced on October 21, 2014. It was an important milestone for a number of reasons. For one, it removed the minimum RAM requirement of 512 MB, allowing devices with less RAM to report themselves as "low-memory". Android Runtime, a new experimental application runtime environment was introduced as a possible (later implemented) replacement for Davlik VM. Improvements to the NFC stack and audio, as well as a new API called the Storage Access Framework, were introduced. Another very important feature was the introduction of the Android Wear platform for smart watches (the versions listed above which contain W).

Version 5.x Lollipop

Unveiled under the codename "Android L" on June 25, 2014, Android 5.0 "Lollipop" became available as official over-the-air (OTA) updates on November 12, 2014. Lollipop's most immediately evident feature is its redesigned user interface, which is built around what became known as "material design". Other important changes include improvements to the notifications, which now can be both visible and interacted with from the lock-screen and also displayed within applications as top-of-the-screen banners. Internal changes include the use of Android Runtime (ART) instead of Dalvik VM. Also, a framework, internally known as Project Volta, which features improvement and optimization of battery usage, is implemented.

The latest major official version, as of June 2015, is Android 5.1, released on March 9, 2015. It features official support for multiple SIM cards, high-definition voice calls between Android 5.1 devices and a different take on device protection: even if reset, a device will remain blocked until its owner signs into their Google account.

A.1.3. Android "ecosystem"

This section briefly describes some further uses for Android, aside from tablets and mobile phones. Many of these "incarnations" include extra features, interface changes and even their own apps in GooglePlay.

Android Wear (`http://www.android.com/wear`)

Android Wear is a trimmed-down version of Android which is used on smart watches and other wearable devices. All of them must pair up with compatible mobile phones, which must use Android 4.3 or later. Google Now technology is integrated and notifications are displayed on the wearable device's screen. Built-in support if offered for searching, messaging, notes, calendar, reminders, alarms, etc.

Since its official release on June 25 2014, hardware manufacturers have responded quite enthusiastically to Android Wear. Big companies which produce devices include Asus, HTC, Intel, LG, Motorola, Qualcomm and Samsung. According to [7], as of February 11 2015, 720000 smart watches were sold. GooglePlay statistics ([3]) show between one million and five million installations, as of June 2015.

The standard features include support for Google Now and the following other apps:

- **Maps**: users can use voice interaction with their phone to find directions and start the journey. As it progresses, the watch will show the directions. In addition, it uses haptic feedback to indicate turns by feel, thereby making looking at the watch face optional.
- **Media and music:** music can be requested using voice input and the volume can also be controlled, both via the touch screen and using voice commands.
- **Fitness:** this feature uses GoogleFit for run and ride tracking, calorie expenditure, steps taken, etc. For compatible devices, heart rate can be shown on the wearable's screen. Cards showing goals reached and other relevant summaries are also displayed.

Android TV (`http://www.android.com/tv`)

Announced on June 25, 2014, Android TV is a smart-TV platform, heralded as the successor to GoogleTV. It offers interactive television experience based on 10-foot user interface. It can be controlled using a mobile phone app, voice commands or with any Android-compatible remote control. Integration is supported as firmware for TVs and stand-alone digital media players alike. The interface is divided into three sections: smart recommendations based on viewing habits, media apps and games at the bottom[1].

Various manufacturers are listed as collaborators, including Razer and Asus, both focusing mostly on games, Sharp, Sony, Philips and TP Vision. The first device to feature

[1]These can be clearly seen on `http://www.android.com/tv`

Android TV was the Nexus Player, co-developed by Google and Asus, and released in November 2014.

Android Auto (`http://www.android.com/auto`)

Announced at the same time as Android TV, on June 25 2014, Android Auto allows mobile devices running Android 5.0 or later to be operated in commercial automobiles using the dashboard's head unit. The standard package offers drivers control over GPS navigation, music playback, SMS, telephony, and web search. The interface can be handled via a touch-screen, even though, for obvious safety reasons, voice communication is highly recommended.

Android Auto is part of the Open Automotive Alliance, a joint effort including around 30 companies, including automobile manufacturers and mobile technology suppliers. Currently, brands such as Alfa Romeo, Audi, Bentley, Chevrolet, Fiat, Ford, Honda, Mitsubishi and Volvo, among others, offer support for Android Auto.

As this section shows, Android is an OS in continuous development and has been deployed on a large number of platforms and devices. It is to be expected that in future it will continue to grow and expand, providing an even more comprehensive list of possible uses.

A.2. Application fundamentals

Android is a complete operating system targeted at mobile devices. This completeness results in a complicated yet well laid out design of applications. This section describes (in varying level of detail) the fundamentals of how an application is built and what it looks like from the inside.

A.2.1. OS considerations

The language Android applications are written in is Java. Other programming languages are possible in principle, but in typical Google style[2], using any of them to program under Android entails complex installation and configuration procedures and is usually prone to errors. Once the code is written, the SDK tools compile the code plus data and resource files into an Android package. This package file bears the suffix `.apk` and all code in it is considered to be one application. Android devices (or the emulator) use this file to install the application.

The Android operating system implements the principle of *least privilege*[3]. This principle states that *for a particular abstraction layer, every module* (process or application in our case) *must be allowed to access only such information and resources that are necessary for its legitimate purpose.* For the case of a multitasking OS, it implies an organization whereby all applications run in independent environments, called *sandboxes,* which include application data, compiled code resources and files which are not available to any other running application (unless explicitly specified otherwise). It is important to note that in practice, true least privilege is neither definable nor possible to enforce. No methods exist today that allow the least amount of privilege for a process to perform its function correctly to be evaluated. It is not possible to know all the values of variables it may process, addresses it will need, or the precise time during which these will be required. The closest practical approach nowadays is to eliminate privileges that can be manually evaluated as unnecessary. Even so, the resulting set of privileges still exceeds the true minimum required privileges.

In the particular case of Android, each application is a different user. By default, the system assigns each application a unique Linux user ID, only known to the system. Permissions are set for all the files in an application so that only the user ID assigned to that application can access them. The *sandboxing* is carried out by making each process run in its own virtual machine, so an application's code runs in isolation from other applications. By default, every application runs in its own Linux process. When any of the application's components need to be executed, Android starts the process. Conversely, processes are shut down when no longer needed or when the system must recover memory for other applications.

[2]Google AppEngine comes to mind, in which, according to Google, any programming language is possible, but in reality the language of choice was Python. Using AppEngine with other languages was so unstable that a frustrated developer once remarked: "Yeah, AppEngine can be used with any language, as long as this language is Python!"

[3]Also known as *principle of minimal privilege* or just *least privilege.*

The result of all this complexity is a very secure environment in which an application cannot access parts of the system for which it is not given permission[4]. Furthermore, applications by default are not permitted to access any data that does not belong to them.

What we just described is the default behaviour. It is possible for an application to share data with other applications and for an application to access system services. For example it is possible to arrange for two applications to share the same Linux user ID. This means they will be able to access each other's files. Also, it can be arranged for applications with the same user ID to run in the same Linux process and share the same VM[5], for example when we want to conserve system resources. An application can request permission to access device data such as the user's contacts, SMS messages, the mountable storage (SD card), camera, Bluetooth, and more. All application permissions must be granted by the user at install time.

A.2.2. Application components

Application components are the essential building blocks of any Android application. Each of them exists in its own right and helps define an application's overall behaviour. Component can be thought of as different points through which the system can enter the application. Not all of them are actual entry points for the user.

There are four different types of application components. Each type serves a distinct purpose and has a distinct life cycle that defines how it is created and destroyed.

Activities

An *activity* represents a single screen with its corresponding user interface. For example, a media player can have one activity that shows a list of available files to play, another activity to reorganize a play list and another activity which contains controls that actually play the chosen files. Activities work together to form a cohesive user experience but each one is independent of the rest. This opens the doors to one of the most interesting, elegant and useful features of Android: if the application allows it, *a different application can start any one of these activities for its own benefit.* For example, a camera application can

[4]As a design feature of operating systems, this idea is not new. For example, *BeOS* implemented a similar sandbox structure system-wide.

[5]For this to happen, the applications must also be signed with the same certificate

start the activity in the email application that composes new mail, to allow the user to share a picture. Programatically, an activity is implemented as a subclass of `Activity.`

Services

A *service* is a component that runs in the background. Services perform longer operations or work for remote processes. Unlike activities, services provide no user interface. A service might play music in the background while the user is in a different application, or fetch data over the network without blocking user interaction with an activity. Services can be started by other components and be allowed to run. Alternatively, a component might bind to a service in order to interact with it. Programatically, a service is implemented as a subclass of `Service`.

Content providers

A *content provider* manages a shared set of application data. Data can be stored in the file system, an *SQLite* database, on the web, or any other persistent storage. Other applications can query or even modify the data using the content provider (if allowed, naturally). As a real example, Android provides a content provider that manages the user's contact information. Any application with the proper permissions can query part of the content provider to read and write information about a particular person. Content providers are also useful for reading and writing private data which is not to be shared. For example, a content provider could be used to store calendars not to be shared with the default Android Calendar, which is in turn shared with Google. Programatically, a content provider is implemented as a subclass of `ContentProvider` and must implement a standard set of APIs that enable other applications to perform transactions.

Broadcast receivers

A *broadcast receiver* is a component that responds to system-wide broadcast announcements. Many broadcasts announce system events such as that the screen has turned off, the battery is low, or a picture was captured. Broadcasts can also be initiated by applications, for example, to let other applications know that some data has been downloaded to the device and is available for them to use. Broadcast receivers have no user interface but they may nonetheless create a status bar notification to alert the user when an event occurs. But more commonly a broadcast receiver is just a "gateway" to other components and is intended to do a very minimal amount of work. Broadcast receivers are implemented as a subclass of `BroadcastReceiver` and each broadcast is delivered as an Intent object.

A unique aspect of the Android system design which deserves special attention is that any application can start another application's component, as mentioned above. The typical example is when we want the user to capture a photo with the device camera. Since this is a very usual action, chances are there is an application on our device which does precisely that. Our application can use this other application, instead of developing an activity to capture a photo. Code does not need to be linked or even incorporated[6]. Instead, we would simply start the activity in the camera application that captures a photo. When complete, the photo is even returned to our application so we can use it. To the user, however, it seems as if the camera is actually a part of the application[7].

When the system starts a component, the process for that application is started, if not already running, and the classes needed for the component are instantiated. In the above example, if we starts the activity in the camera application that captures a photo, it will run in the process that belongs to the *camera application, not in our application's process.* This represents an important difference between Android and most other systems: *applications don't have a single entry point.*

Given that each application is run in a separate process with file permissions that restrict access to other applications, we cannot directly activate a component from another application. *The Android system, however, can.* So, if we want to activate a component in another application, a message must be delivered to the system that specifies our intent to start a particular component. The system then activates said component and allows us to use it.

A.2.3. Activating Components

Activities, services, and broadcast receivers are all activated at run time by an asynchronous message called an *intent*. Intents are the messages that request an action from other components, whether those belong to the application which originates them or to another. Intents are created with an `Intent` object. They can be *explicit* or *implicit*, allowing us to activate a specific component or a specific type of component, respectively.

[6]As a side note, *incorporated* sounds a lot better than *copied and pasted*, doesn't it?

[7]As an example of the difference between using a component versus using a library can be found in section C.2 in which the same action is performed using these two different ways

For **activities** and **services**, an intent defines the action to perform and may also specify the URI of the data to act on. Other data depending on what the component might need to know can also be provided. An intent might, for example, pass on a request for an activity to show an image or to take a photo. Activities can be *invoked for a result*, in which case, the result is returned in an Intent. An application might, for example, issue an intent to let the user pick a personal contact and have it returned to the program, in which case return intent includes a URI pointing to the chosen contact. An activity can be started or given something new to do by passing an `Intent` to `startActivity()` or `startActivityForResult()`, depending on whether we want the activity to return a result or not. Services, on the other hand, can be started or new instructions can be given to an ongoing service by passing an `Intent` to `startService()`. If we wish to bind to a service, we would pass an `Intent` to `bindService()`.

For **broadcast receivers**, the intent simply defines the announcement being broadcast. A typical situation is when the device discovers an open WiFi network. If so configured, it will send a broadcast to indicate that effect. This broadcast will only include a known action string that indicates "Open WiFi network found". From within a program, a broadcast can be initiated by passing an `Intent` to methods like `sendBroadcast()`, `sendOrderedBroadcast()` or `sendStickyBroadcast()`.

Finally, **content providers** are not activated by intents but when targeted by a request from a `ContentResolver`. Content resolvers are an abstraction between the content provider and the component that requests the information or action, left for security reasons. This means that content resolvers work directly with the content provider while the requesting component works with the `ContentResolver` object, executing methods on it. A content provider can be queried by calling `query()` on a `ContentResolver` associated with it.

A.2.4. The Manifest File

Every application must declare various things that it uses, provides and requires to and from the Android OS. This is done in the application's `AndroidManifest.xml` file or the "manifest" file, located at the root of the application project directory. The manifest is an XML file that has the following functions (this list is not exclusive):

- Identification of any user permissions the application requires, for example, Internet access, preventing the device from sleeping or access to the user's contacts.
- To declare all of the application's components.

- To declare the minimum API Level required, based on which APIs the application uses.
- To declare hardware and software features used or required by the application (camera, bluetooth, etc.)
- API libraries the application needs to be linked against, other than the Android framework APIs, such as the Google Maps library.
- And many more

Following are some typical applications of the manifest file.

Declaring components

A manifest file might declare an activity like this:

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

In the `<application>` element, the `android:icon` attribute points to resources for an icon that identifies the application. In the `<activity>` element, the `android:name` attribute specifies the fully qualified class name of the Activity subclass whereas `android:label` specifies a string to use as the user-visible label for the activity.

Most of the file's contents are self-explanatory. We would use:

`<activity>` to declare activities,

`<service>` to declare services,

`<receiver>` for broadcast receivers and

`<provider>` for content providers.

Declaring activities, services, and content providers in the manifest file is absolutely crucial. *Any activity, service or content provider which is implemented in the source code* *but not declared in the manifest can never run.* Broadcast receivers on the other hand can be either declared in the manifest or created dynamically. To use them, we need to registered them with the system by calling the `registerReceiver()` method.

Component capabilities and implicit intents

What was described above is the way intents can be used to start activities, services, and broadcast receivers explicitly. We do that by specifying the target component, using its class name. But the real power and elegance of intents is clearly seen and appreciated when using *intent actions*. Using these is even easier: one has to describe the type of action sought to perform and, if applicable, the data upon which it is to be preformed and *the system itself finds a component on the device that can perform the action and start it.* Multiple components can match this search, in which case the user selects which one to use. The way the system finds a suitable candidate is by looking at the *intent filters* provided in the application manifests of all applications on the device and comparing them to the received intent. Declaring a component in the manifest allows one to include intent filters if so decided. A component's intent filter is declared by adding an `<intent-filter>` element as a child of the component's declaration element.

As an example, suppose we have an application which needs to send an email. It might declare an intent filter in its manifest entry to respond to "send" intents. If we develop another activity, in the same application or distinct and need to use the same functionality, we would create an intent with the "send" action (`ACTION_SEND`), which the system matches to the first application's "send" activity and would launch it when the intent is invoked with `startActivity()`.

Declaring application requirements

There are a variety of devices powered by Android and not all of them provide the same features and capabilities. In order to prevent your application from being installed on devices that lack features needed by your application, it's important that you clearly define a profile for the types of devices your application supports by declaring device and software requirements in your manifest file. Most of these declarations are informational only and the system does not read them, but external services such as Android Market do read them in order to provide filtering for users when they search for applications from their device.

For example, if your application requires a camera and uses APIs introduced in Android 2.1 (API Level 7), you should declare these as requirements in your manifest file. That way, devices that do not have a camera and have an Android version lower than 2.1 cannot install your application from Android Market.

However, you can also declare that your application uses the camera, but does not require it. In that case, your application must perform a check at run time to determine if the device has a camera and disable any features that use the camera if one is not available.

Two features that have to be carefully considered follow.

- Screen size and density: Android splits devices into categories according to the physical dimensions of the screen the physical density of the pixels on the screen, measured in dpi. The screen sizes are: small, normal, large, and extra large whereas the screen densities are: low density, medium density, high density, and extra high density. Unless otherwise specified, a new application is compatible with all screen sizes and densities, given that the Android system makes the appropriate adjustments to the UI layout and image resources. But special layouts for certain screen sizes should be created and special images for certain densities, using alternative layout resources. Next, using the `<supports-screens>` element, we can declare exactly which screen sizes are supported.
- Input configurations: This point refers to the fact that many devices provide different types of user input mechanism. This can be a hardware keyboard, a trackball, or a five-way navigation pad. If a specific type of user input device is required, this can be declared in the manifest with the `<uses-configuration>` element. This is an important feature but it is not typical that an application should require a certain input configuration.

Other device features and their corresponding manifest elements can be found in [17].

A.2.5. Application Resources

Like most GUI applications today, an Android application not just code—it requires additional objects, such as images, audio files and everything else pertaining to the visual presentation of the application. Animations, menus, UI layout of activity, among others, are typically defined using XML files. The collection of all these and other objects used by the application excluding the source code are known as *application resources*. They are a convenient way to update various characteristics of an application without modifying code. They also provide a way to prepare and optimize an application for a variety of device configurations, such as different languages and screen sizes.

Every resource included in an Android project receives a unique integer ID, which can be used to reference it from within the code or from other resources. Resources are presented to the user as instances of the R class. For additional commodity, descriptive identifiers are used in lieu of an integer identifier. For example, an image file named `logo.png`, which would be saved in the `res/drawable/` directory, will receive the resource ID named `R.drawable.logo`, which can be later used to insert the image in the UI.

Aside from an easy way to reference images or other files, another useful application is to define UI strings in XML, thereby providing translations into other languages. Based on the language qualifier of the resource directory's name and the user's language setting, the Android system will find the appropriate language strings and build the UI using them. Still another great feature is creating different layouts. These can depend on the device's screen orientation and size. If the device screen is in portrait orientation (`tall`), a layout with buttons aligned vertically will be more suitable whereas if the screen is in landscape orientation (`wide`), horizontal alignment is more appropriate. We would therefore define two different layouts and apply the appropriate qualifier to each layout's directory name. Then, the system automatically applies the appropriate layout depending on the current device orientation.

B. Development tools for Google Android

Writing applications for Android is no more complicated than programming under any other operating system. Due to the fact that the code will be executed on another machine, the compiling, transfer and execution are more complicated than usual. For this purpose Google have provided an elaborate collection of development tools to aid the development process. This section looks at how to set up the development environment and start using the tools. Firstly, the system requirements are listed, as well as the necessary steps to set up everything up. Secondly, the "bare-bones" development environment using command-line tools is considered. The third part describes the most powerful and convenient way to develop using Eclipse, while the fourth part describes the newcomer to the market and the new recommended way to develop under Android: Android Studio.

Throughout the discussion the development platform is assumed to be Linux. Any flavour of Linux will do, provided it contains the necessary libraries and can run Java. Although the focus of this discussion is the emulator, specific mention will be made on how to develop on a real Android device.

B.1. Android SDK

The heart of Android development is the Android SDK, released by Google as open source software. The SDK provides Android libraries for the different platform versions, an emulator, tools and various samples. Its modular structure allows each version to be individually installed and used. A minimum of one virtual device on which to test is required, even though one can have as many as necessary. If an Android-enabled terminal is present, testing can occur directly on it rather than on a virtual device. Additionally, to aid code development, an IDE can be used.

B.1.1. System requirements

The basic prerequisite to develop under Linux is to have a system which uses GNU C Library `glibc 2.7` or newer. Given that the SDK is written in Java, we also require a reasonably recent version of JDK. [16] lists JDK 5 or JDK 6 as a prerequisite, explicitly stating that JRE alone is not sufficient. The official Sun Java JDK is required, as the GNU Java compiler `gcj` is not supported.

B.1.2. Installing the Android SDK

Once the system requirements are met, the SDK can be installed. To do this, one must grab the source code from `http://developer.android.com/sdk/index.html` Currently, only `.tgz` compressed format is available for Linux. Once downloaded, it can be installed by entering the destination directory and issuing the command

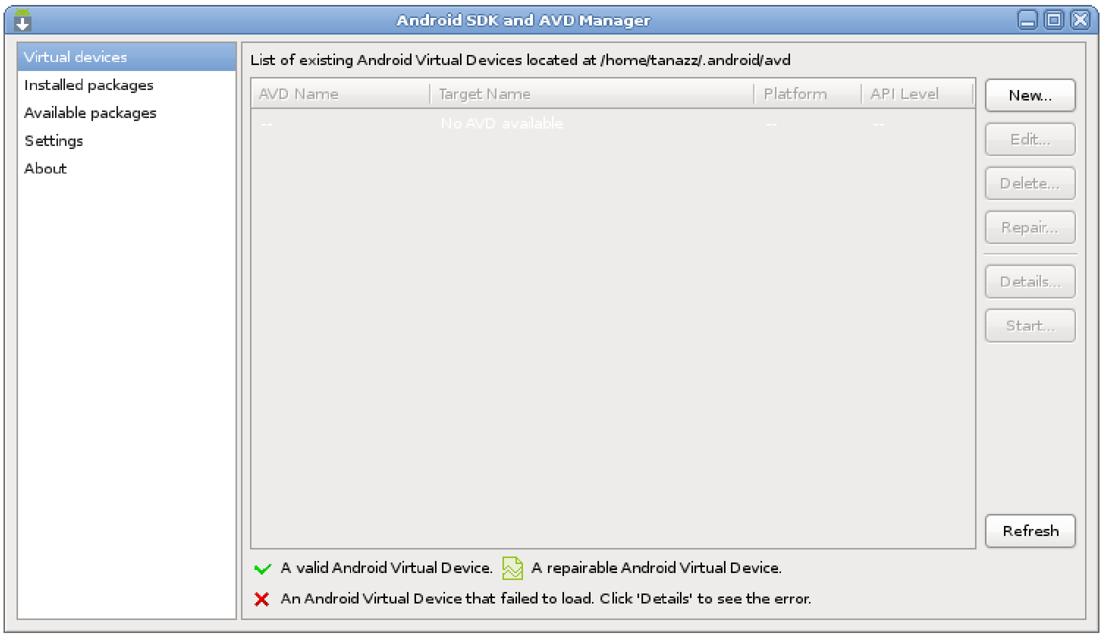
```
tar xvfz android-sdk_r*.tgz
```

which will unpack the SDK to a directory called `android-sdk-<machine-platform>`, where `<machine_platform>` specifies the name of the host hardware platform.

The installed SDK requires approximately 35 MB of disk space for the base SDK, plus 6 MB for the platform-specific tools. This, however, is insufficient to start developing as we have no usable Android platform installed.

B.1.3. Installing an Android platform and additional source code

Any one of the currently supported Android platforms can be installed on our development machine and various platforms can be installed and used side by side. Each of them needs about 150 MB. Developers can optionally install add-ons and samples. Each add-on needs about 100 MB and each set of samples needs about 10 MB. Off-line documentation can also be installed, per platform it needs about 250 MB.



List of existing Android Virtual Devices located at /home/tanazz/.android/avd

AVD Name	Target Name	Platform	API Level
--	No AVD available	--	--

✓ A valid Android Virtual Device. A repairable Android Virtual Device.
✗ An Android Virtual Device that failed to load. Click 'Details' to see the error.

Figure B.1.: Android SDK and AVD Manager run for the first time

Fortunately for the developers, the Android SDK provides a program, aptly named *Android SDK and AVD Manager*, which makes the tedious tasks of managing and downloading the source code (and quite a few others) a breeze. This program can be accessed by entering the `tools/` sub-directory of the SDK installation tree and executing

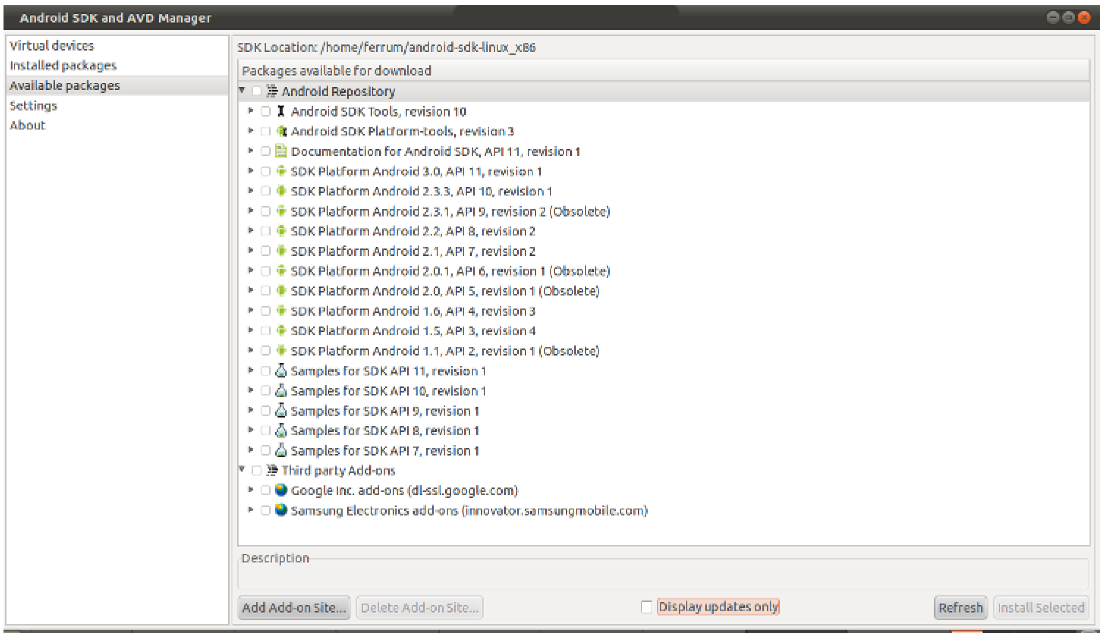
```
./android
```

The result for a freshly installed SDK is shown in figure B.1.

The commands in the left-hand side of the window contain everything we need to set up a new development environment. This is also the central place from where updates, if available, can be installed. At this point the SDK contains only the latest version of the SDK tools, so our next step is to install a platform version. By selecting *Available packages* we can browse the repository and install or update components. An example can be seen in figure B.2.

There are two repositories by default. The *Android Repository* contains the following elements:

- SDK Tools, already installed with the Android SDK, contains everything need for debugging and testing applications, as well as other utility tools. These are located in the `<sdk>/tools/` directory.


Android SDK and AVD Manager

Virtual devices
Installed packages
Available packages
Settings
About

SDK Location: /home/ferrum/android-sdk-linux_x86

Packages available for download

- Android Repository
  - Android SDK Tools, revision 10
  - Android SDK Platform-tools, revision 3
  - Documentation for Android SDK, API 11, revision 1
  - SDK Platform Android 3.0, API 11, revision 1
  - SDK Platform Android 2.3.3, API 10, revision 1
  - SDK Platform Android 2.3.1, API 9, revision 2 (Obsolete)
  - SDK Platform Android 2.2, API 8, revision 2
  - SDK Platform Android 2.1, API 7, revision 2
  - SDK Platform Android 2.0.1, API 6, revision 1 (Obsolete)
  - SDK Platform Android 2.0, API 5, revision 1 (Obsolete)
  - SDK Platform Android 1.6, API 4, revision 3
  - SDK Platform Android 1.5, API 3, revision 4
  - SDK Platform Android 1.1, API 2, revision 1 (Obsolete)
  - Samples for SDK API 11, revision 1
  - Samples for SDK API 10, revision 1
  - Samples for SDK API 9, revision 1
  - Samples for SDK API 8, revision 1
  - Samples for SDK API 7, revision 1
- Third party Add-ons
  - Google Inc. add-ons (dl-ssl.google.com)
  - Samsung Electronics add-ons (innovator.samsungmobile.com)

Description

Figure B.2.: Platform and add-ons selection dialogue

- SDK Platform tools—contains additional tools developed alongside the Android platform. These support the latest features and are updated only when a new platform becomes available. They are contained in `<sdk>/platform-tools/`.
- Android platforms. Each platform component includes a fully compliant Android library and system image, sample code, emulator skins, and version-specific tools, if any. Any production-stage version can be downloaded and deployed on an actual device or used on a virtual device.
- Samples—contains the sample code and applications available for each Android development platform. Highly recommended for first-time users as it contains numerous projects that illustrate many aspects of Android programming.
- Documentation—contains a local copy of the latest documentation for the Android framework API.

The *Third-party Add-ons*, on the other hand, provide components useful when developing application using a specific Android external library (such as the Google Maps library) or a customized Android system image. Additional add-on repositories can be specified by clicking *Add Add-on Site*.

The very basic system consists of the SDK tools, the SDK Platform tools and at least one SDK platform. This is enough to start developing, however, the recommended development system consists of the basic system plus samples and documentation for the installed SDK platform(s).

Once we have decided on what we want to use, we can install the desired elements by selecting them and clicking *Install Selected.* After that, we need to verify and accept any conditions the components may have. At any later point we may check for updates by opening the same window and selecting *Available components* again. By checking the *Display updates only* check-box we can install the elements we do not have.

B.1.4. Structure and uses of a virtual device

We can think of a virtual device as a platform on which to develop and test our application. Internally, however, it is just a number of configuration files for the emulator. These files describe the platform, hardware, software, the optional SD card image to be emulated, etc.

An Android virtual device, or an AVD, consists of the following elements:

- A hardware profile, containing the hardware features of the virtual device, such as the amount of memory, whether or not it has a dialling pad or a physical keyboard, the screen resolution and size, whether it has a camera[1], etc.
- A system image, which defines what version of the Android platform will be run on the virtual device. A choice between the standard Android platform and a system image packaged with an SDK add-on is offered.
- Other options, such as the emulator skin which lets you control the screen dimensions, appearance, etc. You can also specify the emulated SD card to use with the AVD.
- A dedicated storage area on the development machine, in which the device's user data (installed applications, settings, contents of standard folders, etc.) and emulated SD card can be stored.

Internally, the AVD manager, GUI version or otherwise, creates one directory for every AVD and one entry for every file in this directory so it can find them later. It is important to bear this fact in mind when moving the device from its original directory and always do this using the manager (see below).

[1]Unfortunately and inexplicably, the emulator does not emulate the camera in any way

Naturally, as many AVDs as necessary can be created and used. It is recommended to create many devices which to contain different hardware configurations, screen sizes, etc. to test the application thoroughly.

B.1.5. Formatting and using a virtual SD card

A real-world application is more than likely to use the SD card. For this reason, if we want a proper test of our application, we should create a virtual image of the SD card. The program which does the job for us is called `mksdcard` and is located in the `tools/` directory in the SDK tree. The syntax it accepts is

```
mksdcard [-l label] <size> <file>
```

where `label` is the optional name we want to give the card, `size` is by default measured in bytes but can be given in kilobytes or megabytes appending K or M to the numerical value and `file` is the file system name. For example, to create a 128MB SD card image named *SDCard* contained in the file *mySDCard*, one would say

```
mksdcard -l SDCard 128M mySDCard
```

The resulting image is in the IMG format and contains no data.

From here on, we need to access and modify the image. The simplest way is to use the loop-back device capabilities Linux offers and to treat the image as a separate drive. This method offers the convenience that modifying the SD card's contents is as simple as copying or deleting a file from the Linux file system and can be done using any graphical file manager.

Making any file image a part of the Linux file system is called *mounting*. Since we are dealing with a file rather than a physical file system we need to use a loop device[2]. Firstly, we need a directory where to mount the file. Suppose we call the directory *myCard*. To mount the image created above, one would say

```
sudo mount -o loop mySDCard myCard
```

and after providing the administrator's password, we open our favourite file manager and can access and modify the SD card image as if it were a separate drive. When we finish and to make sure the changes are saved, we should unmount the file saying

```
umount myCard
```

[2]A loop device, also called *lofi* (loopback file interface) is a pseudo-device that makes a file accessible as a block device. Mounting a file containing a file system via such a loop mount makes the files within that file system accessible. This is how one can view the contents of CD ISO images without burning the CD.

B.1.6. Creating a virtual device

Using AVD Manager

Once AVD Manager is open, one has to click “New” which opens the window shown in figure B.3.

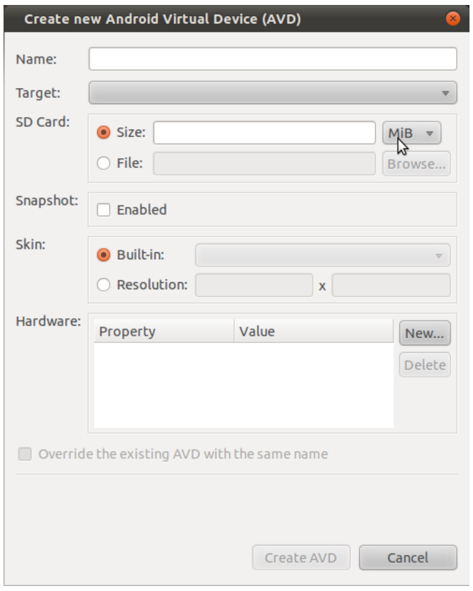


Figure B.3.: New AVD options

Most information in this window is self-explanatory. *Name* sets the name of our AVD, *Target* contains a list of all available platforms and allows us to choose one. *SD Card* gives the programmer a choice: we can either have a blank SD image, choosing the first option, or a previously created (see section B.1.5 for how to do that) SD image, choosing the second option. The *Snapshot* option allows the developer to store the state of the device, providing a service similar to the *hibernate* function on most modern laptops.

The next options are of utmost importance. The *Skin* option specifies the screen size and resolution. It allows the developer to use custom-sized screen (selecting the second option) or a pre-defined screen size (selecting on of the built-in options).

The *Hardware* option opens a window in which we can choose the hardware elements with which to work. It contains a list of important options which are detailed in table B.1.

Characteristic	**Description**
Device ram size	The amount of physical RAM on the device, in megabytes. Default value is "96".
Trackball support	Whether there is a trackball present.
Trackball support	Whether there is a trackball on the device. Default value is "yes"
Touch-screen support	Whether there is a touch screen or not on the device. Default value is "yes"
Keyboard support	Whether the device has a QWERTY keyboard. Default value is "yes"
DPad support	Whether the device has DPad keys. Default value is "yes"
GSM modem support	Whether there is a GSM modem in the device. Default value is "yes"
Camera support	Whether the device has a camera. Default value is "no"
Maximum horizontal camera pixels	Default value is "640"
Maximum vertical camera pixels	Default value is "480"
GPS support	Whether there is a GPS in the device. Default value is "yes"
Battery support	Whether the device can run on a battery. Default value is "yes"
Accelerometer	Whether there is an accelerometer in the device. Default value is "yes"
Audio recording support	Whether the device can record audio. Default value is "yes"
Audio playback support	Whether the device can play audio. Default value is "yes"
SD Card support	Whether the device supports insertion/removal of virtual SD Cards. Default value is "yes"
Cache partition support	Whether we use a /cache partition on the device. Default value is "yes"
Cache partition size	Default value is "66MB"
Abstracted LCD density	Sets the generalized density characteristic used by the AVDs screen. Default value is "160"

Table B.1.: Hardware options and their default values (taken from `http://developer.android.com/guide/developing/devices/managing-avds.html`)

Using command-line tools

If for any reason we wish to create a virtual device using the command line, we need to provide the same information listed above, what differs is the way we would pass it on.

Before we begin, it is a good idea to query the installed platforms on our development machine. We can do this by going to the `<sdk>/tools/` directory and issuing the command
`./android list targets`
which will scan `<sdk>/platforms/` and `<sdk>/add-ons/` for valid system images and return the names of the platforms. An example run of this program follows:

```
Available Android targets:

id: 1 or "android-8"
        Name: Android 2.2
        Type: Platform
        API level: 8
        Revision: 2
        Skins: HVGA, WVGA854, WVGA800 (default), WQVGA432, QVGA, WQVGA400
id: 2 or "android-10"
        Name: Android 2.3.3
        Type: Platform
        API level: 10
        Revision: 1
        Skins: HVGA, WVGA854, WVGA800 (default), WQVGA432, QVGA, WQVGA400
id: 3 or "android-Honeycomb"
        Name: Android Honeycomb (Preview)
        Type: Platform
        API level: Honeycomb
        Revision: 1
        Skins: WXGA (default)
```

Notice that every different platform installed has an id and a name by which it is referenced internally (the line `id:  2 or "android-10"`).

To manipulate an AVD from the `tools/` directory, one must use the familiar command `./android` with a sub-command specifying what is to be done. Thus, the basic syntax to create an AVD is

```
./android create avd -n <name> -t <targetID>
```

where `targetID` is the internal id of the platform. So if we want to create an AVD named *Test* which uses the Android 2.3.3 in the above configuration, we would say

```
./android create avd -n Test -t 2
```

The program next asks us if we want to create a custom hardware profile. The default answer is *no* and if we just hit *Enter*, the device will have the following default hardware profile:

Abstracted LCD density: 240
Heap size for the virtual machine: 24KB
RAM size for the device: 256MB

If on the other hand our answer is *yes*, the program will ask us to set values to every one of the configuration parameters listed in table B.1, just as if we were doing this from the GUI.

By default, the program creates the AVD file in the `~/.android/avd/` directory[3]. If we want to specify a different one, we may do so with the `-p` parameter like so

```
./android create avd -n Test -t 2 -p /path/to/avd/folder
```

B.1.7. Renaming, moving, deleting and editing AVDs

The best way to edit an AVDs configuration is using AVD Manager and choosing the option `Edit`. Deleting AVDs is also best done through the GUI, even though the command

```
android delete avd -n <name>
```

will easily accomplish the task. When renaming or moving, the best option is to use the command line tools. The command

```
android move avd -n <name> -p <path>
```

will move the AVD to `path`, while

```
android move avd -n <name> -r <newName>
```

will rename `<name>` to `<newName>`.

Important note: it is crucial to refrain from performing these actions changing just the AVDs file names or deleting them. As we discussed in section B.1.4, the internal structure of an AVD is complex and is managed by the SDK. Therefore, we are best off manipulating the AVDs using the SDK tools.

[3]As with all Linux distributions, the ~ symbol is a shortcut to the user's home directory

B.1.8. Configuring Android enabled devices

With an Android-powered device, we can develop and debug Android applications just as we would on the emulator. There are, however, a few things to configure before we can do that.

1. The application must be declared as "debuggable" in the Android Manifest (see ??). We do this by adding `android:debuggable="true"` to the `<application>` element.
2. The "USB Debugging" option must be turned on on the device. This is accomplished by going to the home screen, pressing MENU, selecting `Applications > Development`, then `Enable USB debugging`.
3. The last step is to set up the development system to recognize the Android device[4]. Developing on Linux, requires us to add a rules file that contains a USB configuration for each type of device to be used for development. Each device manufacturer uses a different vendor ID. Example rules files and the vendor IDs can be found at `http://developer.android.com/guide/developing/device.html`. Executing adb devices from the `platform-tools/` directory will assure us that the devices is connected. If everything went correctly, we will see the device name listed as a "device".

If using Eclipse (see below), running or debugging happen as usual. A Device Chooser dialogue will be presented that lists the available emulator(s) and connected device(s). We then select the device upon which we want to install and run the application. Finally, if the Android Debug Bridge (`adb`) is used, commands can be issued with the `-d` flag to target the connected device.

B.2. Developing using command-line tools and adb

Contrary to what it might seem, Android command-line tools are a great and fun way to develop and run applications. Using them will also provides insight into how the whole system functions.

The whole developing process revolves around three basic tools. The `android` tool, examples of whose use we have presented, is used to create and manipulate the structure

[4]Funnily enough, the Android Developers Website states regarding this point: "If you're developing on Mac OS X, it just works"

of project files. `Apache Ant`[5] is the tool which compiles the source code into `.apk` file which is then transferred and executed on the emulator using `adb`.

B.2.1. Creating projects

To create an Android project, the `android` tool is used. It generates a project directory with some default application files, stub files, configuration files and a build file. The syntax to achieve this is as follows:

```
android create project --target <target_ID> --name <your_project_name> \
--path path/to/your/project --activity <your_activity_name> \
--package <your_package_name-space>
```

A brief description of this somewhat confusing command line follows.

target is the build target for the application. It corresponds to an Android platform library (including any add-ons, such as Google APIs), as specified by the `android list targets` command (see section B.1.6).

name is the name of our project. It is optional and if provided, will be used for the `.apk` file name when the application is built.

path is the location of the project directory. If the directory does not exist, it will be created.

activity is the name for the default Activity class and also the `.apk` file name if a project name is not specified.

package is the package name space for the project. It follows the same rules as any Java package.

Once created, a peek in the project's directory reveals which files need to be modified and worked on.

B.2.2. Building and running projects

There are two ways to build applications under Android. One is for testing or debugging, aptly named *debug mode* and one when building the final package for release, called *release mode*. No matter which method is used, the application has to be signed before it can

[5] Apache Ant is a software tool for automating software build processes. It is similar to Make but is implemented using the Java language, requires the Java platform, and is best suited to building Java projects. Unlike Make, which uses Makefiles, Ant uses XML files.

install on an emulator or device—with a debug key when in debug mode and with our own private key when building in release mode.

The building process happens using the `Ant` tool, which will create the `.apk` file that will install on an emulator or device. In debug mode, the `.apk` file is automatically signed by the SDK tools with a debug key, so it's instantly ready for installation onto an emulator or attached development device. An application signed using a debug key cannot be distributed. In release mode, the resulting `.apk` file is unsigned, so it has to be signed manually with the developer's private key, using `Keytool` and `Jarsigner`.

Once we are ready to run our application, we navigate to the root of the project directory and say

```
ant debug
```

If everything goes fine, the next step is to install the compiled `.apk` file onto the emulator and run it. We navigate to the `tools/` directory and issue the command

```
adb install <path/to/project/bin/subdir>.apk
```

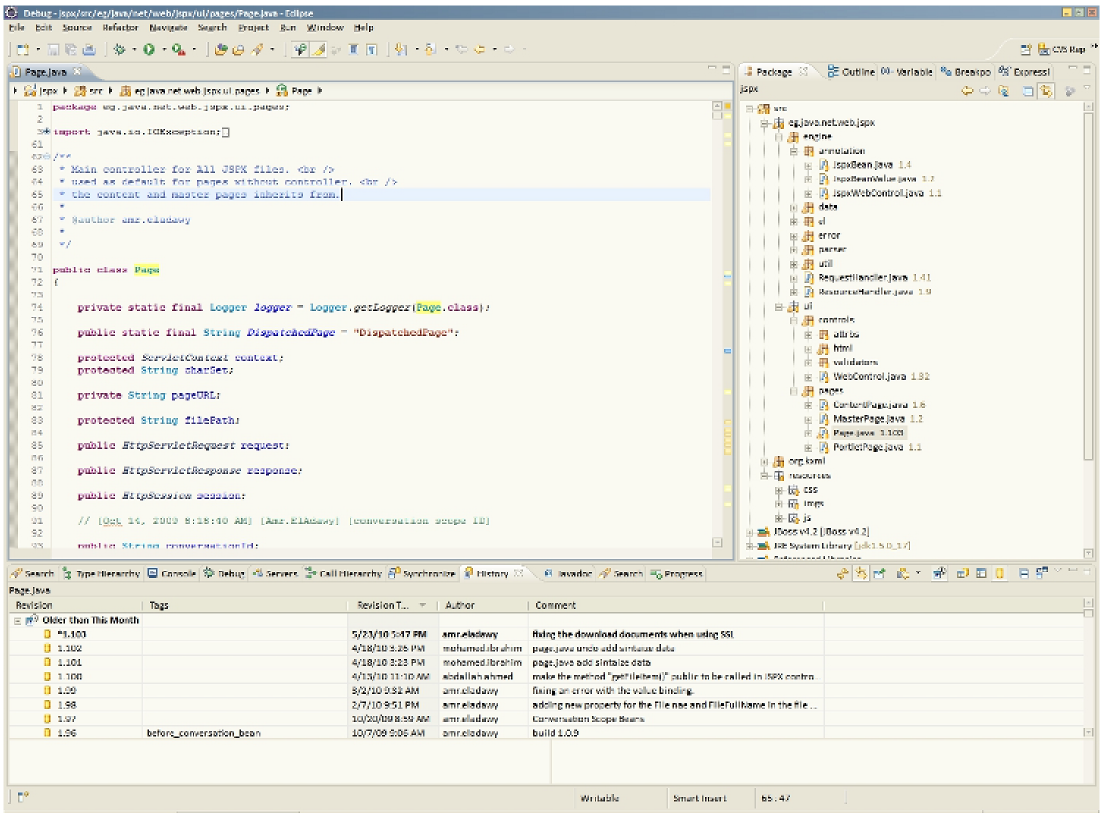
B.2.3. Debugging applications

While ADB has numerous powerful options to debug any application and change its state to see how it might respond, a look [?] at its description proves it to be excessively complicated. In section B.3.5 we describe the recommended way to debug applications. Notice that ADT uses ADB extensively but it would be infeasible for a user to work directly with ADB if a good profiling and debugging is to be performed in a reasonable amount of time.

B.3. The tried-and tested way: Eclipse and ADT

As we saw, one can develop under Android using just command-line tools. No IDE is required but anyone who has developed software consisting of many classes and other sorts of files can definitely vouch for the usefulness of IDEs. The concept of grouping all files an application uses into what is known as a project file, conveniently visualizing and editing them as necessary and the ease with which the application can be built and debugged are all great commodities and time savers. A developer will certainly appreciate the order IDEs bring to the dozens of classes and thousands of lines of code. Android is no exception.

In many programmers minds the phrase "programming IDE" is synonymous with Eclipse. Highly modular and actively developed, Eclipse is much more than an IDE. Its core is written mostly in Java. It employs what is known as lightweight component framework to which is owes its great additional functionality. Except for its small size lightweight kernel, everything else is structured as a plug-in. This vastly extensible and flexible plug-in system allows Eclipse to be used to develop in many other programming languages including Ada, C, C++, COBOL, Perl, PHP, Python, Ruby (including Ruby on Rails framework), Scala, Clojure, and Scheme. Further, Eclipse can, again using plug-ins, be used with typesetting languages like LaTeX, XML and XSL, networking applications such as telnet and SSH, and a variety of database management systems. Some of the most popular extensions include powerful GUI builders and class flow diagram creators. Currently, over a thousand plug-ins are in active development [23]. A screen-shot of Eclipse in action is shown in figure B.4 [29].


Debug - jspx/src/eg/java/net/web/jspx/ui/pages/Page.java - Eclipse
File Edit Source Refactor Navigate Search Project Run Window Help
Page.java
jspx ▸ src ▸ eg.java.net.web.jspx.ui.pages ▸ Page ▸
Package Outline Variable Breakpo Expressi
Search Type Hierarchy Console Debug Servers Call Hierarchy Synchronize History Javadoc Search Progress
Writable Smart Insert 65 : 47

```
package eg.java.net.web.jspx.ui.pages;

import java.io.IOException;

/**
 * Main controller for All JSPX files. <br />
 * used as default for pages without controller. <br />
 * the content and master pages inherits from.
 *
 * @author amr.eladawy
 *
 */

public class Page
{

    private static final Logger logger = Logger.getLogger(Page.class);

    public static final String DispatchedPage = "DispatchedPage";

    protected ServletContext context;
    protected String charSet;

    private String pageURL;

    protected String filePath;

    public HttpServletRequest request;

    public HttpServletResponse response;

    public HttpSession session;

    // [Oct 14, 2009 8:18:40 AM] [Amr.ElAdawy] [conversation scope ID]

    public String conversationId;
```

Page.java

Revision	Tags	Revision T...	Author	Comment
Older than This Month				
*1.103		5/23/10 5:47 PM	amr.eladawy	fixing the download documents when using SSL
1.102		4/18/10 3:26 PM	mohamed.ibrahim	page.java undo add sintaize data
1.101		4/18/10 3:23 PM	mohamed.ibrahim	page.java add sintaize data
1.100		4/13/10 11:10 AM	abdallah.ahmed	make the method "getFileItem()" public to be called in JSPX contro...
1.99		3/2/10 9:32 AM	amr.eladawy	fixing an error with the value binding.
1.98		2/7/10 9:51 PM	amr.eladawy	adding new property for the File nae and FileFullName in the file ...
1.97		10/20/09 8:59 AM	amr.eladawy	Conversation Scope Beans
1.96	before_conversation_bean	10/7/09 9:06 AM	amr.eladawy	build 1.0.9

Figure B.4.: Eclipse in action

Eclipse features under heavy development (and very popular these days) include the Server platform and the Web Tools platform. The former allows for remote server programming, debugging and interaction with applications running on the attached server. The stunning possibility to install the required server for development *directly from the graphical interface* is also offered. The latter feature contains tools for developing Web and Java EE applications. It includes source and graphical editors for a variety of languages, wizards and built-in applications to simplify development, and tools and APIs to support deploying, running, and testing web-apps. Last but not least, Eclipse has been translated into more than a dozen natural languages.

All these plug-ins and added functionality are available to Eclipse users through the use of repositories, similar to Linux distributions today. Content in these repositories can be signed or unsigned, coming from trusted or untrusted sources.

As far as Java is concerned, Eclipse has a powerful code completion engine which can identify to which class objects, functions and methods belong and import said class automatically with a single keystroke. It features the Java Development Tools with a built-in incremental compiler which allows for substantial reduction in compile time. Because of this technique, Eclipse can spot syntax errors immediately and suggest possible solutions, allowing syntax and (some) semantic errors to be corrected with a single keystroke.

All this taken into account, it is not surprising that Google should state Eclipse as their IDE of choice. Not only that, Google actually *recommend* developing using Eclipse. For this reason, the ADT plug-in was created. ADT stands for Android Development Tools. It greatly simplifies the initial configuration, code management, installation, running, logging and debugging. One can say that every part of the development process can be completed with the ADT plug-in easily, quickly and without the need to manipulate any configuration files by hand, thus enabling the developer to concentrate on the application and not on the internals.

This section looks at the tasks already described in section B.2 but done from within Eclipse. Special attention is paid to the manipulation of the manifest file and the design and testing of the GUI. We assume that Eclipse has been installed, for tips on how to do that you can visit `http://www.eclipse.org`

B.3.1. Eclipse workspace and perspectives

Eclipse's interface may seem very intimidating at first, so before plunging into it, it might help to clarify two of the basic concepts behind the design of its GUI, namely work-spaces and perspectives. This will help us later on to understand how to move between the various processes in the development cycle of the applications.

One can think of a *workspace* as a designated area where various projects and their data files are stored and manipulated. In the simplest case, this is just a directory, typically in the user's home folder. Eclipse also supports mounted work-spaces on network drives, `rsync` or `ftp` through the appropriate plug-ins. If so configured, Eclipse can work with multiple work-spaces simultaneously.

In figure B.4 we can see an example of the user's view of the workspace in Eclipse[6]. The three most usual and easily recognizable parts of a workspace are the file editor (in the top left-hand side), which is where the editing process happens, the workspace browser in the top right-hand side which contains the various files, packages, variables, class hierarchy, etc. and the console tab in the bottom part of the screen, which contains tools for debugging, error and/or warning messages, class property editor and `javadoc` messages, among others. This is the initial or default workspace that contains everything one needs to write and correct code.

A *perspective* on the other hand is an arrangement of tools, options and commands depending on their use. It is a convenient way to group different parts of Eclipse and make them visible and usable when necessary. For example, the debug perspective may contain an editor for variables, stack watch, breakpoints, error messages and program output, all these in a different screen from the development screen described above. A CSS perspective might provide tools for colour selection, table, banner and page formatting, separated from the design of the actual web page it may apply to. In figure B.5 there is an example of another very useful perspective. Strictly speaking, the workspace view described above is just one of the various perspectives, grouping the tools necessary for editing and running code[7] and keeping the rest aside until we need them.

Perspectives can be created by the user. A good tutorial on how to create them and how they work internally can be be found at `http://www.eclipse.org/articles/using-perspectives/PerspectiveArticle.html`

B.3.2. Installing and configuring the ADT plug-in

System requirements

ADT, as expected, requires a working installation of Eclipse, version 3.5 (Ganymede) or greater. Given that Eclipse is written in Java, a working installation of JRE, JDK

[6]Quite possibly your development environment looks different from this screen-shot, the position of the individual elements varies from one version to the next

[7]In Eclipse it is called the *Java Perspective* if the development language is Java or by the name of the programming language of the project otherwise

5 or 6 is also a must but these tend to be installed together with Eclipse. Once again, the Android Development Team explicitly state that the open-source implementation compiler `gcj` is not supported and only the official Java packets have to be installed.

Installation process

The easiest way to install ADT is using Eclipse itself. From the *Help* menu one selects *Install New Software...* and clicking *Add* in the top right-hand corner. The relevant fields in the dialogue which follows should be filled in like this:

Name:	`ADT Plug-in`
Location:	`https://dl-ssl.google.com/android/eclipse/`

After hitting the *OK* button and when the next screen appears, in the *Available Software* dialogue, we select *Developer Tools* and click *Next*. Clicking *Next* in the next window and accepting the license agreement concludes the installation. If a security warning is issued, we dismiss it pressing *OK*. When the installation is complete, Eclipse must be restarted.

Configuration

Once ADT is safely installed on the computer, the next step is to modify the ADT preferences in Eclipse to point to the Android SDK directory. We do so by selecting *Window–Preferences* and selecting *Android* from the left panel. A question might be asked on the next screen if we want to send usage statistics to Google. Whatever our answer, we cannot continue until we click *Proceed*.

Next, we click *Browse* and locate our SDK directory. The configuration process is completed by clicking *Apply*, then *OK*.

Updates

These are the steps to take when we want to update the plug-in. Firstly, we have to select *Help-Check for Updates*. If there are no updates available, a dialogue will inform you of that. If there are updates, we select *Android DDMS, Android Development Tools*, and *Android Hierarchy Viewer*, and click *Next*. In the next dialogue, *Update Details,* we click *Next*. The update process is concluded by accepting the license agreement and clicking *Finish*. To use the new version, we should restart Eclipse.

B.3.3. Creating projects

The ADT plug in provides the *New Project Wizard* with which a new empty project can be created or we can import existing code. To run the wizard, we select *File–New–Project*. After that, we select *Android–Android Project*, and click *Next*. We next enter a name for our project, which is also the name of the folder where it will be stored. Under *Contents*, we select *Create new project in workspace* and select the location of our workspace. Under *Target*, we have to select an Android target to be used as the project's Build Target. This specifies which Android platform the project will be built against. It is recommended to select the lowest platform with which our application is compatible. Note that this setting can be changed at any time by right-clicking the project in the *Package Explorer*, selecting *Properties–Android* and then checking the desired *Project Target*.

Under *Properties*, we fill in all necessary fields. *Application name* is the human-readable title for the application as it will on the Android device. *Package name* is the package name-space, which follows the same rules as any Java package. All our code will reside in that name-space. *Create Activity* is optional, but recommended and it can contain the name for our main Activity class. We need to select a *Min SDK Version.*, indicating the minimum API Level required for the application to run properly. This automatically sets the `minSdkVersion` attribute in the `<uses-sdk>` of the application's *Android Manifest* file. If in doubt, a good recommendation is to use the API Level listed for the *Build Target* in the *Target* tab. Clicking *Finish* completes the creation of a new project.

B.3.4. Building and running projects

By default, whether we use an actual device or the emulator, the build process constantly runs in the background as the project is changed. During this automatic build, Eclipse enables debugging and signs the `.apk` file with a debug key. When the application is run, Eclipse invokes ADB and installs it to a device or emulator, automatically performing all the same tasks listed in section B.2.2 so that the programmer does not have to worry about parameters, commands and syntax and can concentrate on writing code. It is important to note that if we want to distribute the application, the `.apk` file must be signed with our private key.

Run configurations

The run configuration is an optional (but useful) set of parameters that specify the project to run, the Activity to start, the emulator or connected device to use, etc. When the project is run for the first time, a run configuration will be created. By default, the default project Activity will be launched and automatic target mode for device selection will be used. If this is not what we want, we can always modify the run configuration or create a new one.

The *Run* configuration manager is accessible from the *Run* menu. After that, we expand the *Android Application* item and can create a new configuration or open an existing one. There are various settings to be configured. The project and activity to be launch are in the Android tab. The Target tab allows us to choose between Manual or Automatic mode to select an AVD to run the application on (see next paragraph for a detailed description). Additionally, parameters can be specified to the emulator via the Additional Emulator Command Line Options field. For example, the `-scale 96dpi` to scale the AVDs screen to an accurate size, based on the computer monitor. A full list of emulator options can be found at `http://developer.android.com/guide/developing/tools/emulator.html`

Automatic and manual target modes

A run configuration will by default use the automatic target mode in order to select an AVD on which to run the application. The following rules apply:

- If there's a device or emulator already running and its AVD configuration meets the requirements of the application's build target, the application is installed and run on it.
- If there's more than one device or emulator running and all of them meet the requirements of the build target, a dialogue is shown prompting the user to select which device to use.
- If no devices or emulators are running that meet the requirements of the build target, ADT looks for a match to the build target in all available AVDs. If one is found, ADT chooses that AVD.
- If no suitable AVDs are found, the application is not installed and a console error warning tells us there is no existing AVD that meets the build target requirements.

If, however, a "preferred AVD" is selected in the run configuration, the application will always be deployed to that AVD. A new emulator will be launched if it's not already running.

The manual mode simply presents a "device chooser" every time the application is run and we can select on which AVD to install and run it.

Running the application

Running or debugging an application is done by selecting *Run–Run* or *Run–Debug.* ADT automatically creates a default run configuration for the project (see above) and compile the project, if there have been changes since the last build. Next, it installs and starts the application on an emulator or device, depending on the run configuration. By default, Android run configurations use an "automatic target" mode for selecting a device target (see above). If the chosen option is *Run*, Eclipse then installs the application on the device and launches the main activity. If we chose *Debug* instead of *Run*, the application will start in the *Waiting for debugger* mode and once it has attached, Eclipse will open the *Debug perspective* and will only then start the application's main activity.

If we want to develop on a device, there is no significant difference in the steps to take. However, we must make sure the device has been properly set up as explained in section B.1.8.

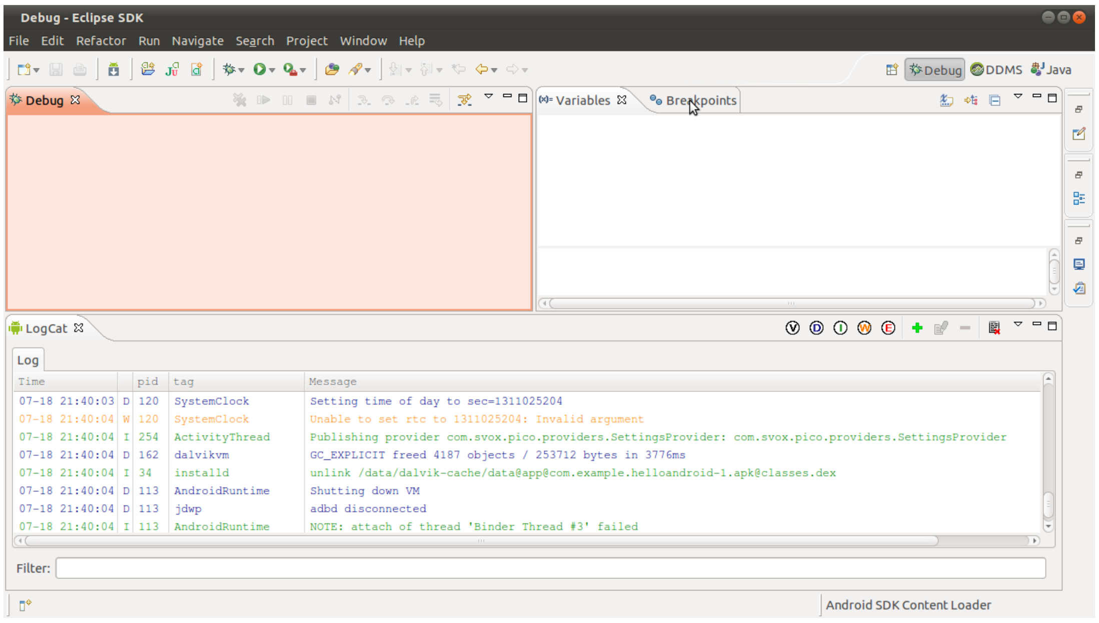
B.3.5. Debugging from within ADT

Debugging is a vital process in application development so it is no surprise that ADT has excellent tools to that end. A presentation of the two ways to debug an application follows. The reader will no doubt realize the degree of ease ADT brings in this regard when compared to the command-line debugger.

Eclipse's Debug perspective

Eclipse provides the possibility to debug any application project using its own built-in debugger. It can be accessed through *Window–Open Perspective–Debug.* A screen-shot is shown in figure B.5.

Four tabs deserve attention. The *Debug* tab shows Android applications and their currently running threads being debugged. *Variables* displays variable values during code


Debug - Eclipse SDK
File Edit Refactor Run Navigate Search Project Window Help
Debug DDMS Java
Debug
Variables
Breakpoints
LogCat
Log
Filter:
Android SDK Content Loader

```
Time           pid  tag              Message
07-18 21:40:03 D 120  SystemClock      Setting time of day to sec=1311025204
07-18 21:40:04 W 120  SystemClock      Unable to set rtc to 1311025204: Invalid argument
07-18 21:40:04 I 254  ActivityThread   Publishing provider com.svox.pico.providers.SettingsProvider: com.svox.pico.providers.SettingsProvider
07-18 21:40:04 D 162  dalvikvm         GC_EXPLICIT freed 4187 objects / 253712 bytes in 3776ms
07-18 21:40:04 I 34   installd         unlink /data/dalvik-cache/data@app@com.example.helloandroid-1.apk@classes.dex
07-18 21:40:04 D 113  AndroidRuntime   Shutting down VM
07-18 21:40:04 D 113  jdwp             adbd disconnected
07-18 21:40:04 I 113  AndroidRuntime   NOTE: attach of thread 'Binder Thread #3' failed
```

Figure B.5.: Eclipse Debug Perspective

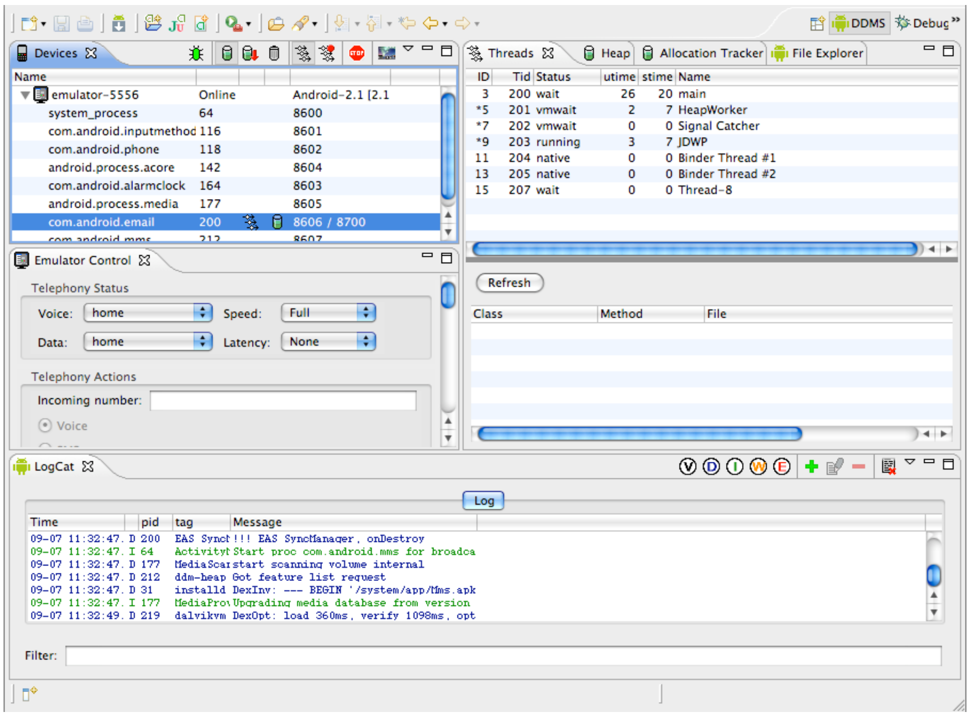
execution. Notice that just like any other debugger, this perspective only works properly when breakpoints have been set. *Breakpoints* displays a list of the set breakpoints. *LogCat*, also available in the DDMS perspective, allows us to view system log messages in real time.

DDMS perspective

DDMS or Dalvik Debug Monitor Server is a debugger extension shipped with Android. The main debugger is still adb, DDMS combines various ADB instances and enables the user to interact with them using DDMS rather than with each one individually. Additional functionality DDMS brings is screen capture on the device, port forwarding, radio state information and spoofing, threads and heap information and a lot more. This section describes the most relevant parts of the DDMS perspective and what they can be used for.

We can get to the DDMS perspective on Eclipse from *Window–Open perspective–Other–DDMS*. A screen-shot of an arrangement that contains its most relevant elements is shown in figure B.6.

Before we explain the elements in the screen-shot, it may help to examine how the DDMS works with the debugger. As explained in section A.2, every application on Android runs



Name				
emulator-5556	Online			Android-2.1 [2.1
system_process	64			8600
com.android.inputmethod	116			8601
com.android.phone	118			8602
android.process.acore	142			8604
com.android.alarmclock	164			8603
android.process.media	177			8605
com.android.email	200			8606 / 8700

ID	Tid	Status	utime	stime	Name
3	200	wait	26	20	main
*5	201	vmwait	2	7	HeapWorker
*7	202	vmwait	0	0	Signal Catcher
*9	203	running	3	7	JDWP
11	204	native	0	0	Binder Thread #1
13	205	native	0	0	Binder Thread #2
15	207	wait	0	0	Thread-8

Voice: home Speed: Full

Data: home Latency: None

Incoming number:

Voice

Class	Method	File

```
Time             pid  tag      Message
09-07 11:32:47. D 200  EAS Synct!!! EAS SyncManager, onDestroy
09-07 11:32:47. I 64   ActivityI Start proc com.android.mms for broadca
09-07 11:32:47. D 177  MediaScarstart scanning volume internal
09-07 11:32:47. D 212  ddm-heap Got feature list request
09-07 11:32:47. D 31   installd DexInv: --- BEGIN '/system/app/Mms.apk
09-07 11:32:47. I 177  MediaProvUpgrading media database from version
09-07 11:32:49. D 219  dalvikvm DexOpt: load 360ms, verify 1098ms, opt
```

Filter:

Figure B.6.: DDMS perspective(taken from [22])

in its own virtual machine. A part of this virtual machine is its own copy of ADB which runs internally and provides a unique external port to which a debugger can attach. When DDMS starts, it connects to ADB. Every time a new VM (i.e. a new application) is started or terminated, ADB is notified and it in turn notifies DDMS, which connects directly to the VM's profiling port and can thus talk directly to the VM's copy of ADB. This connection and the data transfer thereafter are accomplished through the `adbd` daemon.

DDMS assigns a debugging port to each VM on the device, starting (by default) with port 8600 for the first debuggable and incrementing the port number by 1 every time another instance is run. When a debugger connects to one of these ports, all traffic is forwarded to it from the associated VM. Only one debugger can attach to a single port but DDMS can handle multiple attached debuggers.

Another DDMS port is of importance, number 8700. It is a forwarding port which accepts traffic from any VM on the device and forwards it to port 8700, thus enabling all VM's to be debugged from a single port one at a time. The traffic is determined by the currently selected process in the DDMS Devices view (see figure B.6).

We now describe how to perform some common debugging tasks. A more complete discussion can be found in [22]. Note that there is no distinction between a physical device and a virtual device when using DDMS.

To work with an emulator or device's file system, one turns to the *File Explorer* tab which contains commands to view, copy, and delete files on the device. This can come handy when we need to examine files created by the application or if we want to transfer files to and from the device. To **copy a file from the device**, we locate the file and click the *Pull file* button. To **copy a file to the device**, we click the *Push file.*

To **view the log messages** an application issues using the *Log* class or other logged system messages such as stack traces, we turn to the *LogCat* feature. The various types of messages can be filtered choosing their type or creating more complex filters, for example warning and error messages filtered by process id.

To emulate phone operations and location, one turns to the *Emulator control* tab, where options to simulate a phone's voice and data network status can be found. This helps to test an application's robustness in differing network environments. The *Telephony Status* section of the *Emulator controls* tab manages different aspects of the phone's networks status, speed and latency. Changes to these are effective immediately.

To spoof calls and messages, the *Telephony Actions* section of the *Emulator controls* tab is of help. This is useful for testing an application's robustness in responding to incoming calls and messages.

To set a (mock) location, one turns to the *Location Control* tab. This is useful for testing different aspects of the application's location specific features without physically moving. A variety of geolocation data types are available.

For further information on Eclipse, the `www.eclipse.org` website is the best source. As for ADT, there is not much documentation to be consulted and so probably the best way to learn it and to see what it can do to simplify a programmer's life is to experiment with its features.

B.4. The newcomer and Google's recommendation: Android Studio

Android Studio (`https://developer.android.com/tools/studio/index.html`) is the newest contender in the field. Built by Google on top of IntelliJ IDEA, it is the official IDE for Android development [2]. Its first preview version, 0.1, was available in May 2013 while the first stable version, 1.0, was released in December 2014.

This section will briefly describe the new features and a few of the differences that can be found with ADT. It does not mean to provide a full and complete description given that a lot of what was said about ADT is still applicable, the differences are mostly "cosmetic".

B.4.1. Build framework: Gradle

Instead of Apache Ant, Android Studio uses Gradle as its building framework. Gradle is a build automation tool which supports substantial multi-project builds. A distinctive feature is its incremental build, which Gradle accomplishes by adaptively determining which parts of the build tree need to be updated, thus speeding up build process by not re-building the whole tree. Instead of the XML description of the project tree favoured by Apache Ant, Gradle uses a Groovy-based domain-specific language [5] for this purpose.

B.4.2. Android-specific quick fixes

Given that this IDE was specifically written for Android, code refactoring and quick fixes are specific to Android and arguably much more useful. Code refactoring is an important tool for making the code more legible, more compact and therefore, easier to maintain; all this without having any impact on the code's external behaviour. An excellent treatment of commonly used refactoring techniques can be found in [14]. For a Java-specific take on the same topic, [10] is an excellent source.

Quick fixes aim to offer ways to correct frequently made coding mistakes, once those are spotted by the IDE. For example, if a programmer attempts to use a class without including its parent package, a quick fix will be suggested which will include said package; where there is more than one package possible, a list will be presented so the developer can choose which one to include. Another frequently encountered mistake is trying to use a function which throws an exception without a `try` and `catch` block; in this case, a quick fix will be suggested which will surround the code with a `try-catch` block or alternatively, the function which this code is a part of will be declared to throw this exception. The idea is to correct these mistakes so that the compiler will be able to go through the code without any problems, thereby speeding up the whole process.

B.4.3. Lint tools

In programmer lingo, *lint* refers to certain pieces of code which, without being syntactically incorrect, are likely to cause bug-like behaviour. For example, using a variable without declaring it first or using constants in conditional blocks which make the condition constant, possibly leading to unintended infinite loops. Lint tools are very common in all IDEs and those included in Android Studio detect performance, language and API-compatibility issues, among others.

B.4.4. GUI layout editor

Creating a GUI is a much more straightforward process with Android Studio, thanks in no small part to the fact that different orientations, resolutions and screen sizes can be easily visualised. In addition, the XML editor that came standard in ADT is included and within easier reach. A screen shot of this feature, taken from `http://www.filehorse.com`, is shown in figure B.7. A related feature is the wizard-like interface to create common Android components, views and designs. Last but not least, the drag-and-drop features of ADT are expanded upon and more components are readily available.

B.4.5. Other features

There are many other additions to Android Studio as compared to Eclipse and ADT. Among those worth mentioning are:

- support for the Android Wear platform
- off-the-shelf support for Google Cloud, enabling integration with apps hosted on Google Cloud
- built-in signing capabilities using ProGuard. While ADT also offered this functionality, Android Studio provides it seamlessly as a part of the deployment process
- improvements to the visual coding interface and real-time app rendering

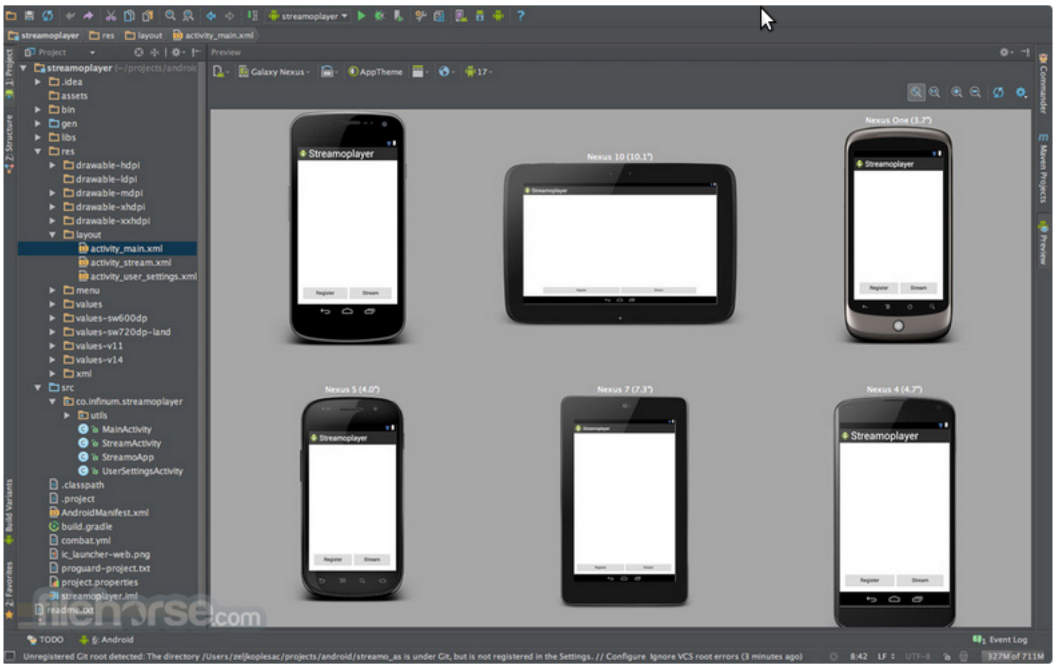


Figure B.7.: Android Studio's GUI layout editor

- easier-to-set-up run configurations
- more straightforward handling of virtual devices
- improvements in the debugging process and console, both using virtual and physical devices

As a conclusion, this appendix provided a somewhat historical account of the ways to develop under Android. All four basic ways were used at different points in the development process for this project which is why it was deemed necessary to include all of them, however outdated some might seem nowadays, in this appendix.

C. ZXing internals

The following sections describe ZXing from a programmer's perspective, focusing on more "exotic" features such as its installation from source and using it to generate QR codes. Some some code snippets showing its usage form within the application are shown.

C.1. Installing the ZXing library on Android

There are two basic ways to do that. One can download the `.zip` archive, build it and install it onto the phone. The alternative and recommended way is to install Barcode Scanner directly, either via the `.apk` package provided or from Android Market.

Building ZXing

To do this, aside from the source code for the library, one must also have *ProGuard* present on their computer. By default, it should be a part of the Android SDK but in case it is not there, it should be installed. *ProGuard* shrinks, optimizes, and obfuscates Java code by removing unused code and renaming classes, fields, and methods with semantically obscure names. The result is a smaller sized `.apk` file that is more difficult to reverse engineer. It should be used when the application uses features that are sensitive to security.

To build ZXing, the first step is to grab the `.zip` archive from the project website, `http://code.google.com/p/zxing/`. Next, we edit the `build.properties` file at the top level of the project, changing the `android-home` property to point to where our SDK is installed. Also, we should set the `proguard-jar` property to the full path, file name included, of the *ProGuard* library. Having done this, we can start the building process. ZXing is built without debugging symbols to prevent conflict between *ProGuard* and the Android tool chain. We have to navigate to the unpacked directory and say

```
cd core

ant clean build-no-debug
```

Next, we build the Android code issuing the commands

```
cd ../android

ant
```

With the device connected and set up for installation as described in section B.1.8, we navigate to `bin/` and find the `BarcodeScanner-debug.apk` file. We install it by saying

```
adb install
```

Our library is now installed on the phone and can be used to scan codes as described in sections C.3 and C.4.

Installing the package file

This is the recommended way to install ZXing as it takes away the complexity of the build and install process. The easiest way is to find the Barcode Scanner application in Android market and install it from there. Alternatively, one can grab the `BarcodeScanner.apk` file and install it by simply saying

```
adb install BarcodeScanner.apk
```

C.2. Obtaining an image or a preview from a camera

QR codes rely on images, so naturally, some attention must be paid on how we can obtain an image from the device's camera.

From within an application

Before we start, our application has to declare its intentions to use the camera and set the correct permissions. This is done in the Android Manifest file, in particularly the following three lines.

```
<uses-permission android:name="android.permission.CAMERA" />
<uses-feature android:name="android.hardware.camera" />
<uses-feature android:name="android.hardware.camera.autofocus" />
```

Auto-focus has to be enabled for the camera to find the QR code properly. It also helps if we try and fit only the QR code in the viewfinder.

To access the camera, we need to:

1. Instantiate the Camera class first. We can do that by calling the open() method. This will return an object whose parameters we might have to set.
2. We check the returned settings with `getParameters()`. If necessary, they can be modified using the `Camera.Parameters` object. To set the new parameters, we call `setParameters(Camera.Parameters)`.
3. If desired, we can set the display orientation using `setDisplayOrientation(int)`.
4. The camera needs a fully configured `SurfaceHolder` to start the preview. We can do this by calling `setPreviewDisplay(SurfaceHolder)`.
5. Preview must be started before we can take a picture. `startPreview()` keeps updating the preview surface.
6. When we see fit, `takePicture(Camera.ShutterCallback, Camera.PictureCallback, Camera.PictureCallback, Camera.PictureCallback)` will capture a photo.
7. After taking the picture, preview would have stopped. We have to wait for the callbacks to provide the actual image data.
8. If we wish to take more photos, `startPreview()` must be called again first. `stopPreview()` will stop updating the preview surface.
9. When finished, it is important to call `release()` to allow the camera to be used by other applications.

This is just the orientative list of actions that need to be taken in order to take a picture from within our application. These steps are recommended by the Android Developers. There is, however, a far less complicated way to do that.

Using an intent

As [24] suggests, using an intent provided by MediaStore is the standard way to obtain a picture from the camera. This method uses a temporary file to store the photo and then it is just read from there when the intent ends. The following code snippet, taken from [24] illustrates this idea. Note that this code should be run from within an Activity.

```
private static final int TAKE_PHOTO_CODE = 1;

private void takePhoto(){

  final Intent intent = new Intent(MediaStore.
     ACTION_IMAGE_CAPTURE);
  intent.putExtra(MediaStore.EXTRA_OUTPUT, Uri.fromFile(
     getTempFile(this)) );
  startActivityForResult(intent, TAKE_PHOTO_CODE);

}

private File getTempFile(Context context){

  //this returns /sdcard/image.tmp

  final File path = new File( Environment.
     getExternalStorageDirectory(), context.getPackageName()
     );

  if(!path.exists()){
    path.mkdir();
  }

  return new File(path, "image.tmp");
}

protected void onActivityResult(int requestCode, int
   resultCode, Intent data) {

  if (resultCode == RESULT_OK) {

    switch(requestCode){
      case TAKE_PHOTO_CODE:

        final File file = getTempFile(this);
        try {
          Bitmap captureBmp = Media.getBitmap(
             getContentResolver(), Uri.fromFile(file) );

         // here the image can be manipulated as necessary
         // (resized, converted to greyscale, etc.)

         // we don't care about the exceptions, so just
         // catch them and ignore them
        } catch (FileNotFoundException e) {
          e.printStackTrace();
        } catch (IOException e) {
          e.printStackTrace();
        }
      break;
    }
  }
}
```

C.3. Calling ZXing from within a program

Calling ZXing in a program requires an instance of `com.google.zxing.Reader`. An implementation that can detect all formats the library reads can be used. In this case, we would proceed like so:

`Reader reader = new MultiFormatReader();`

Alternatively, if we know exactly the kind of code we are reading, we can instantiate an implementation that only understands this particular kind of code. This is the more efficient method. So, for QR Codes,

`Reader reader = new QRCodeReader();`

The next step is the image to decode. Readers receive as parameters instances of `com.google.zxing.MonochromeBitmapSource`. These are just abstractions on top of various classes representing images, presenting them as a monochrome image to be decoded. There is an implementation for instances of `java.awt.BufferedImage`:

```
BufferedImage myImage = getImageFromCamera(); // dummy procedure
LuminanceSource source = new BufferedImageLuminanceSource(myImage);
BinaryBitmap bitmap = new BinaryBitmap(new HybridBinarizer(source));
```

And now, to decode,

```
Result result = reader.decode(bitmap);
```

`com.google.zxing.Result` has a number of methods that give access to the raw bytes or text encoded by the barcode found. It will also reveal key points in the image related to the barcode that was found, such as the three finder patterns in a QR Code and the bottom-right alignment pattern. The following code snippet illustrates some of these, with self-explanatory variable names:

```
String text = result.getText();
byte[] rawBytes = result.getRawBytes();
BarcodeFormat format = result.getBarcodeFormat();
ResultPoint[] points = result.getResultPoints();
```

Finally, the decoders support a system of "hints" that help the decoder work more efficiently, or trade off accuracy for speed when appropriate. For example, the "TRY_HARDER" hint asks the decoders to spend much more time searching for a barcode:

```
Hashtable<DecodeHintType, Object> hints = new Hashtable<DecodeHintType, Object>();
hints.put(DecodeHintType.TRY_HARDER, Boolean.TRUE);
Result result = reader.decode(bitmap, hints);
```

C.4. Decoding via Intent

Intents are a convenient way to call certain functions from programs already installed on the platform. If an intent is provided, it could be called from any program just as if it was a part of it rather than a module in another package. Therefore, because of its simplicity, this is the recommended way to scan codes when only *BarcodeScanner* is installed as opposed to ZXing package. The following code snippet shows how to scan a code via and intent. It assumes there is a button and when it is pressed, the intent is activated.

```
public Button.OnClickListener mScan = new Button.
   OnClickListener() {
    public void onClick(View v) {
        Intent intent = new Intent("com.google.zxing.client.
           android.SCAN");
        intent.setPackage("com.google.zxing.client.android");
        intent.putExtra("SCAN_MODE", "QR_CODE_MODE");
        /* Optionally, this is where one would put the hints.
         * For example, to insert a TRY_HARDER hint, one
           would say
         * intent.putExtra("DecodeHintType.TRY_HARDER", "true
           ");
         */
        startActivityForResult(intent, 0);
    }
};

public void onActivityResult(int requestCode, int resultCode,
    Intent intent) {
    if (requestCode == 0) {
        if (resultCode == RESULT_OK) {
            String contents = intent.getStringExtra("
               SCAN_RESULT");
            String format = intent.getStringExtra("
               SCAN_RESULT_FORMAT");
            // Scan successful, proceed
        } else if (resultCode == RESULT_CANCELED) {
            // Cancelled, proceed
        }
    }
}
```

C.5. Decoding codes from a web page

Starting with version 3.3, *BarcodeScanner* can be instructed to scan from a web page and have the result returned to your site via a callback URL. This is done by linking to a URL and properly escaping the parameter's value. For example,

```
http://zxing.appspot.com/scan?ret=http://foo.com/products/{CODE}
/description&SCAN_FORMATS=QR
```

The `ret` parameter specifies the URL to call back with the scan result. `{CODE}` may appear anywhere and will be replaced with the scanned barcode contents. `SCAN_FORMATS` may be optionally used to supply a comma-separated list of format names.

This option is handy when we develop software which has to scan QR codes from web-pages and an Internet connection is present. The alternative is to have our application scan through the HTML source code and decode each image file individually, thus calling ZXing on the phone and using up battery. The server call back method uses the Internet connection and does not place any strain on our device's processor.

C.6. Using ZXing to create QR codes

There is a web-based interface to the ZXing library that can be used to create QR codes. It is located at `http://zxing.appspot.com/generator/` and features some of the preferred formats of information QR codes can store. After the information is input and we hit *Generate*, the resulting code appears in a new page and can be downloaded and used as required.

For a more automatic way of generating these codes, it is suggested to turn to *Google Chart Tools*[1], which has special modules to create QR codes.

[1]It can be found at `http://code.google.com/apis/chart/`

Bibliography

[1] About QR codes. Online. http://www.denso-wave.com/qrcode/aboutqr-e.html.

[2] Android Studio Overview. Online. https://developer.android.com/tools/studio/index.html.

[3] Android Wear - Android Apps on GoogGoogle. Online. https://play.google.com/store/apps/details?id=com.google.android.wearable.app.

[4] Code 39 overview. Online. http://www.racoindustries.com/barcodegenerator/1d/code-39.aspx.

[5] Gradle website. Online. https://gradle.org.

[6] IntelliJ IDEA. Online. https://www.jetbrains.com/idea/.

[7] Just 720,000 Android Wear smartwatches shipped last year. Online. http://www.engadget.com/2015/02/11/android-wear-2014-shipments/.

[8] Oracle documentation for Java. Online. https://docs.oracle.com/javase/7/docs/technotes/guides/jni/.

[9] Q1 2014: QR Code Trends. Online. http://www.qrstuff.com/blog/2014/04/02/q1-2014-qr-code-trends.

[10] Refactoring Java code. Online. http://www.methodsandtools.com/archive/archive.php?id=4.

[11] Symbol versions for QR codes. Online. http://www.denso-wave.com/qrcode/qrgene2-e.html.

[12] "Tu Salario" website. Online. http://www.tusalario.es.

[13] Using Camera API - Marakana. Online. http://marakana.com/forums/android/examples/39.html.

[14] What is Refactoring? Online. http://c2.com/cgi/wiki?WhatIsRefactoring.

[15] Android Developers. Installing the SDK. Online. http://developer.android.com/sdk/installing.html.

[16] Android Developers. System Requirements. Online. http://developer.android.com/sdk/requirements.html.

[17] Android Developpers. Application Fundamentals. Online. http://developer.android.com/guide/topics/fundamentals.html.

[18] AppBrain. Free vs. paid Android apps. Online. http://www.appbrain.com/stats/free-and-paid-android-applications.

[19] AppBrain. Number of Android applications. Online. http://www.appbrain.com/stats/number-of-android-apps.

[20] Bulgarian Tourist Association. The 100 National Tourist Sites (in Bulgarian). Online. http://100nto.org.

[21] Adams Communications. Specifications For Popular 2D Bar Codes. Online. http://www.adams1.com/stack.html.

[22] Android Developers. Using DDMS. Online. http://developer.android.com/guide/developing/debugging/ddms.

[23] Eclipse Team. Eclipse Marketplace homepage. Online. http://marketplace.eclipse.org/.

[24] Tutorial for Android Website. Take pictures in Android with ACTION_IMAGE_CAPTURE. Online. http://www.tutorialforandroid.com/2010/10/take-picture-in-android-with.html.

[25] Margalit Fox. Alan Haberman, who ushered in the bar code, dies at 81. *The New York Times*, June 2011.

[26] Gartner.com. Gartner Says Smartphone Sales Surpassed One Billion Units in 2014, 2015. www.gartner.com.

[27] Prince McLean. Canalys: iPhone outsold all Windows mobile phones in Q2 2009. Online. http://www.appleinsider.com/articles/09/08/21/canalys_iphone_outsold_all_windows_mobile_phones_in_q2_2009.html.

[28] Steve Ranger. "iOS versus Android. Apple App Store versus Google Play: Here comes the next battle in the app wars". Online. http://www.zdnet.com/article/ios-versus-android-apple-app-store-versus-google-play-here-comes-the-next-battle-in-the-app-wars/.

[29] Alex Ryazastev. Eclipse 3.6 Helios Preview Screenshot. Online. `http://upload.wikimedia.org/wikipedia/commons/a/a2/Eclipse_3.6_Helios.jpg`.

[30] Martin Schoeberl. Hardware Support for Embedded Java. Online. `http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.228.1436&rep=rep1&type=pdf`.

[31] The Eclipse Development Team. Using Perspectives in the Eclipse UI. Online. `http://www.eclipse.org/articles/using-perspectives/PerspectiveArticle.html`.

[32] Wikipedia. Android version history. Online. `https://en.wikipedia.org/wiki/Android_version_history`.

[33] Wikipedia. Data Matrix. Online. `http://en.wikipedia.org/wiki/Data_Matrix`.

Nomenclature

ASCII	American Standard Code for Information Interchange
AVD	Android Virtual Device
CAS	Computer Algebra System
cross-compilation	the process of creating executable code for a platform other than the one on which the compiler is running
DDMS	Dalvik Debug Monitor Server
ESSID	Extended Service Set Identification, used in infrastructure-based wireless networks. Also referred to as SSID, it is the identifier of the wireless network
GUI	Graphical User Interface
HTML	Hyper Text Markup Language
IDE	Integrated Development Environment
ISBN	International Standard Book Number
ISO	International Organization for Standardization
ISSN	International Standard Serial Number
JDK	Java Development Kit
JIT	Just-In-time
JNI	Java Native Interface
JRE	Java Runtime Environment
NFC	Near-field communitcation

OCR	Optical Caracter Recognition
OS	Operating System
RFID	Radio Frequency Identification
SDK	Software Development Kit
symbol	In optical(scannable) codes, a *symbol* is the graphical representation of the encoded information.
UI	User Interface
URI	Uniform Resource Identifier
URL	Uniform Resourse Locator
VAT	Value Added Tax
VAT	Value Added Tax
VM	Virtual Machine
VoIP	Voice over IP
VPN	Virtual Private Network
XML	Extensible Markup Language

Index