A Cross-Platform Health Care Application

RICKARD ERIKSSON, SAJJADALI HEMANI
Abstract

A Cross-Platform Health Care Application

Rickard Eriksson, Sajjadali Hemani

This report describes the development process of the mobile application HealthyWay. The application is developed as a Bachelor thesis project together with KTH Royal Institute of Technology. The applications’ main function is to be the interface to a number of wearable medical electronic appliances such as bandages and pill boxes. The communication is done through Bluetooth Low Energy. The application was mainly written in 3 different languages, QML, C++ and Java. The most stressed point throughout this project was the importance of platform independence. The flexibility had to be done on a scalable as well as on a modular level. With this in mind, QT was used to address the requirement of scalability. The modularity was achieved through an organized structure to the program accompanied by easy directives to introduce new devices. The purpose of the application is to increase the quality of life for the user. This intent could however not be concluded due to the lack of testing among users. In its current state, the application is not yet ready for deployment but instead acts as a framework for future prospects.

Keywords: QT, Scalable, Modular, Bluetooth Low Energy, Wearable electronics, Health care, Mobile application, Java, C++, QML, Cross-platform, Android, iOS
Sammanfattning
A Cross-Platform Health Care Application
Rickard Eriksson, Sajjadali Hemani


Nyckelord: QT, Skalbar, Modulär, Bluetooth Low Energy, Wearable electronics, Health care, Mobilapplikation, Java, C++, QML, Multiplattform, Android, iOS
Contents

List of Figures vi
List of Tables vii
Listings viii
Acronyms ix

1 Introduction 1
  1.1 Background ..................................................... 1
      1.1.1 HealthyWay ................................................ 1
  1.2 Problem ........................................................ 2
  1.3 Purpose ......................................................... 2
  1.4 Goal ............................................................. 3
  1.5 Method .......................................................... 3
  1.6 Delimitations .................................................... 4
  1.7 Outline .......................................................... 4

2 Background 5
  2.1 Past work ......................................................... 5
  2.2 The HealthyWay project ......................................... 5
  2.3 Wearable Electronics ............................................ 6
  2.4 Bluetooth Low Energy ........................................... 6
      2.4.1 The GATT profile explained ............................... 7
      2.4.2 BLE in HealthyWay ......................................... 8
  2.5 The hardware used with the application ..................... 8

3 Method 9
6.4 The future of QT and HealthyWay .............................................. 27
6.5 Other areas of application ......................................................... 28

Bibliography .................................................................................... 29

A HEALTHY WAY, Bluetooth LE Protocol Specification V2.0 .............. 32
# List of Figures

2.1 The HealthyWay project ........................................ 6  
2.2 A figure of the GATT profile [17] .......................... 7  
4.1 Design overview for JNI ..................................... 14  
4.2 Hierarchy of HealthyWay ................................. 15  
5.1 Flowchart of the application when pressing the ”Scan” button ....... 19  
5.2 The design of the application GUI ........................... 20  
5.3 A basic picture of what has to be done to add a characteristic .... 24  

List of Tables

3.1 Search results from KTH Primo [26-03-2015] .................. 10

4.1 Criterion for the development platforms .......................... 12
## Listings

5.1  Connection code from dataPage.qml ................................. 18
5.2  Example of updateData from healthywayfunctions.cpp .................. 21
5.3  Native C++ from jnihealthyway.cpp .......................... 22
# Acronyms

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>BLE</td>
<td>Bluetooth Low Energy</td>
</tr>
<tr>
<td>QML</td>
<td>Qt Meta Language or Qt Modeling Language</td>
</tr>
<tr>
<td>XML</td>
<td>Extensible Markup Language</td>
</tr>
<tr>
<td>iOS</td>
<td>iPhone Operating System</td>
</tr>
<tr>
<td>GATT</td>
<td>Generic Attribute Profile</td>
</tr>
<tr>
<td>UUID</td>
<td>Universally Unique Identifier</td>
</tr>
<tr>
<td>ECG</td>
<td>Electrocardiogram</td>
</tr>
<tr>
<td>GUI</td>
<td>Graphical User Interface</td>
</tr>
<tr>
<td>API</td>
<td>Application Programming Interface</td>
</tr>
<tr>
<td>HTML</td>
<td>HyperText Markup Language</td>
</tr>
<tr>
<td>CSS</td>
<td>Cascading Style Sheets</td>
</tr>
<tr>
<td>JNI</td>
<td>Java Native Interface</td>
</tr>
</tbody>
</table>
Chapter 1

Introduction

Wearable electronics is a category within electronics with a focus on portability where the device provides simple services. The application was developed to work together with wearable electronics using the Bluetooth Low Energy protocol. The application is a part of a bigger project called HealthyWay which aims to experiment with the possibility of using wearable electronics to monitor health. The electronics discussed in this report provide primarily medically related services to the user. Applications with similar goals are available however, the scope of this project was to make the program as general and scalable as possible. With that in mind, a lot of focus was given to ease the transition between multiple platforms. The desire for scalability was addressed through the program structure by minimizing the effort to add compatibility with new devices. Throughout this report the recurring focus around scalability and generality is made as that is the main focus of this project.

1.1 Background

The operating system market for mobile devices is closest to an oligopoly where Google and Apple are the notable players [26]. As of the first quarter of 2015, Google’s Android and Apple’s iOS made up 96.3% of the market [26]. Given this very obvious divide in popularity, the project was directed towards mainly these two operating systems.

The chosen development platform was QT (pronounced "cute") which is a contender for being one of the most popular cross-platform application frameworks. The platform was developed in 1994 and is today part of the subsidiary Digia Plc [12].

1.1.1 HealthyWay

The project consists of two main parts, one of the parts is the smartphone application discussed in this thesis and the other part is the hardware that the application communicates with.
The project and application was named HealthyWay and will from now on be referred to by this name. It was carried out as a bachelor thesis at the School of ICT. The thesis is part of a collaboration between Astra Zeneca and KTH. Fredrik Jonsson at KTH Royal Institute of Technology served as the supervisor for this project and for the purpose of the working method assumed the role of the stakeholder.

The hardware that served as the device was from the company Nordic Semiconductor [14]. The product that was used for this project was the nRF51822 [15]. From this device, the heart rate service was broadcasted which in turn was accessed by the HealthyWay application. The printed patch is meant to be placed on the human body to monitor the well-being of the user with different sensors and the application in this thesis is supposed to show the user the output of these sensors.

1.2 Problem

An area where wearable electronics has the potential to play a major role is in the world of health care. As the article from Wired [19] states, a lot of research is being conducted in this field. Information about a patient can prove to be invaluable and potentially save lives. To monitor the well-being of a patient and the effectiveness of a treatment the patient either has to go to the doctor, or the doctor will have to go to the patient. In some cases it could suffice with a phone call, but in most cases it does not. Today there are a lot of development going on around wearable electronics that can communicate through Bluetooth low energy.

1.3 Purpose

The purpose with this report is to give a clear picture of how the smartphone application for HealthyWay works. The report should explain in such detail that anyone that wants to develop the product further should be able to refer to this document for guidance. The thesis explains what cross-platform is and what tools are necessary. The application itself is described in detail, what parts that are modular, what is needed to change different parts of the program and the general structure of the application.

From a more abstract point of view the purpose with the project is to make it easier for hospital personnel and an individual to monitor a certain medical treatment. This could help the hospital personnel considerably because they could monitor the patient remotely which could lead to better treatments. Maybe the finished product could be implemented in other areas of wearable electronics, not just for medical purposes.
1.4 Goal

The goals for this project were decided upon together with the supervisor who also took on the role of a stakeholder. As the supervisor was technically proficient it eased the process.

- **Goal 1**: A major goal for this project was to learn about the possibilities of wearable electronics. In addition to this, to test their compatibility, in a limited sense, with the Bluetooth Low Energy protocol.

- **Goal 2**: On top of having an application which provides a quality of life upgrade, the scalability of the program is given equal focus in this project. By doing this, it is easier to introduce the application to another platform and hence making it readily available to a larger audience.

- **Goal 3**: Together with the goal of scalability a modular structure was desired. This has more to do with the structure of the program. Apart from making the program extendable across platforms, the ease with which a new device could be made compatible with the application was given a high priority. This is to promote further development of the application and to make future patching easier.

- **Goal 4**: To go with all of this a well made documentation is to be made to serve as a manual for future developers. This documentation is to serve as an API for reference where the functions are explained as well as clear directives are given in order to add platform or device compatibility.

Throughout this report the goals will be alluded to as it will be explained how they have been met. A direct reflection will also be done in the conclusion section.

1.5 Method

When delving into scientific processes the major segregation that is made is between inductive and deductive processes [8]. Both showcase strength and weaknesses and are used befittingly.

A deductive process is typically seen as a more theoretical approach to a problem where conclusion are made based on reasoning rather than empirical data. A deductive proof is by definition assumed to be certain. In terms of programming, this method is suited for studies that work closely with computer science.

Inductive reasoning has its roots in empirical results where some sort of experiment has been carried out. An important distinction to be made from the aforementioned deductive reasoning is that inductive proof is never claimed to be certain but rather suggests a recurring pattern. In terms of programming this could be applied to performance evaluations when analyzing algorithms.
It should be noted that when developing this application no traditional method could be directly applied as they are more suited for scientific research whereas this was more of a practically rooted project.

1.6 Delimitations

One of the goals of the project was to have a cross-platform product that would work with the majority of modern operating systems. But because of the limited time and knowledge of the students there had to be certain limitations. A major limitation is that this project will only cover the Android platform because of its ease to work with. Another delimitation that had to be made was the details of the graphical interface. The product has a basic design but has to be improved and tested.

1.7 Outline

Chapter 2 gives a background to the problem areas of the project. Chapter 3 describes the different methods that were used during the project. Chapter 4 gives a description on how the cross-platform development works. Chapter 5 presents the product in detail and how to expand it. Chapter 6 discusses the result of the project. Chapter 7 gives the conclusion of the whole project.
Chapter 2

Background

In the following section, the theoretical background to the HealthyWay application is broken down and explained in greater detail.

2.1 Past work

This application was developed as the final link in a chain, where a significant amount of groundwork has been done prior to this project. A notable project that could be seen as the basis for the application was "Wireless Hybrid Bio-Sensing with Mobile based Monitoring System" [27] by Linlin Xu.

Alongside a number of homepages comparing the different platforms, a useful bachelor thesis was found which aided the decision making process [10].

2.2 The HealthyWay project

Advances in wearable electronics made this project possible as the application was designed with these devices in mind. Flexible electronics and ultra-low-power radio transmitter enable devices to include sensors that can be worn without much discomfort. When considering this, the unobtrusiveness of these devices is important as that is a determining factor for the user [28]. For the scope of this thesis, only the ECG sensor is considered and demonstrated, however, the idea is to have several devices interacting with the application. In Figure 2.1 a simple diagram illustrating a potential future setup is shown in which the HealthyWay application plays an important role. A future prospect for this system would be to incorporate the collected information with medical personnel to deliver greater care.
2.3 Wearable Electronics

The devices that the HealthyWay application is made to interact with are mainly wearable electronics. With the most outstanding quality being the portability, the area of application is immense. By being as hidden as possible while at the same time interacting with the surroundings, wearable electronics are mainly made to improve the quality of life of the user. Popular examples of wearable electronics are Google Glass [3] and Apple Watch [2]. Along with their portability they impose other challenges that must be considered. Aesthetics as well as the level of pervasiveness they have on the user are two obstacles which creators of these devices must overcome. Without considering these two areas it becomes increasingly difficult to convince users to purchase the product. The market for wearable electronics is rapidly growing and specifically taking large strides in the area of medicine [19]. Sensors for measuring movements and vital signs can be used to provide a better picture of the user’s health. This in turn can also be used to deliver better health care as the collected data can be invaluable.

2.4 Bluetooth Low Energy

Bluetooth Low Energy or better known as Bluetooth Smart is a more power efficient version of the classical Bluetooth protocol. The power efficiency is achieved through more primitive modulation techniques [4]. Similar to the classical Bluetooth framework, Bluetooth Low Energy makes use of generic attribute profiles, or GATT to implement profiles. To get a better picture of the hierarchy see Figure 2.2. A profile is a definition of a device for a specific application. In order to introduce compatibility for a new device the GATT must be implemented correctly. A profile typically contains a number of services from which the connected users can obtain data. A level under a profile, several services can be implemented which in turn contains the characteristics. On an interface level, it is the characteristics that are displayed to the end user as they contain the interesting information. From a programming point of view however, each hierarchical level must be dealt with in a proper manner.
2.4.1 The GATT profile explained

As mentioned before, a profile contains at least one service. A service [9] is used as a collection of characteristics belonging to the same category. For example a service named ”Device Information” could hold characteristics such as the name of the device manufacturer, serial number and model number. Each service has its own address called UUID which serves as a unique identifier.

Characteristics also have UUIDs to identify them. It is the characteristics [9] that holds the data that the user can use. The characteristics comes in a wide variety, some of them are predefined by Bluetooth SIG [16] others are defined by independent programmers. A characteristic can be everything from the name of the manufacturer to a steady stream of ECG data. To access the data held within the characteristics there are different ways. One
way is to read the value inside directly, a characteristic that uses this is the battery level. Another way is to use *notify*, by using this the device connected to the BLE device receives a steady stream of the data. The *notify* feature is great to use with real time data like ECG or respiration readings. Besides reading the data on the BLE device it is also possible to write to the characteristics. This is very useful in a situation where a characteristic needs to be activated or deactivated.

### 2.4.2 BLE in HealthyWay

The HealthyWay application has a large dependency on the Bluetooth as it is the chosen way of communication to the devices. For this project Bluetooth Low Energy was very suitable as efficient power usage was a major concern. The wearable electronics, for practical reasons, have a limited amount of space for the electronics which restrains the battery time of the devices. To fully utilize the battery, BLE seemed suitable.

### 2.5 The hardware used with the application

In parallel with this thesis, another thesis [24] concerning the hardware was done by another student. The point of that project was to write the code for the hardware to measure different medically related parameters like ECG and respiration. A BLE profile was built around this to send the results of the readings to the HealthyWay application in this thesis. The profile developed followed a predefined specification in Appendix A which was used to develop the HealthyWay application as well.
Chapter 3

Method

This chapter tells what type of methods that were used during the project. This includes the methods used for research as well as the methods used for the development of the product.

3.1 Research Methodology

Before the chosen method for this project is discussed an explanation of the contesting alternatives is done in order to strengthen the discussion.

As this project revolves around a product, a lot of practical skills had to be applied which led to a significant separation, in terms of time, between theory and development. The importance of planning in the form of pseudo-code was instead escalated which took the role of the theory in this project. Due to the priority put on modularity and scalability, the structure of the program was critical. In order to achieve an optimal structure for the program a hierarchy was designed followed by a discussion as to what each level should encompass. This discussion in turn contributed to the discourse regarding the choice of a development platform. This is discussed in greater detail below.

Another potential approach to this project was to do a more formal background study to help outline the project. With the correct level of granularity of the research, a lot of hurdles could have been avoided. Another advantage to the background study is that it could have partially served as a formal documentation to the program. The drawback of this method was related to the time constraint, the working method and the nature of the project. Doing a detailed background study would take up a lot of time which would be an obstacle for achieving the goals. As mentioned earlier, the project was more rooted in practical skills which shaped our general approach to solving problems. However, with the time factor removed it would have been beneficial in almost every aspect to have done a detailed theoretical study.

If the goal had been to research around the possibility of making the application instead of actually developing the product the technical study would have been a much bigger part
of the project. Although this was a practical project a lot of research had to be done in the form of reading APIs and forums for help regarding implementations. All the options during the development of the project, such as the development tools, were considered and compared to pick the most beneficial method possible.

3.1.1 Information retrieval

Due to the practical nature of the project, the most helpful sources were usually API documentations along with discussion forums. However, when choosing the development platform a more formal literature study was carried out. The primary information source for this was the KTH ”Primo” source along with Google searches.

![Search results from KTH Primo](26-03-2015)

<table>
<thead>
<tr>
<th>Search term</th>
<th>Search results</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cross-platform</td>
<td>11,253</td>
</tr>
<tr>
<td>Cross-platform + tools</td>
<td>2,481</td>
</tr>
<tr>
<td>Cross-platform + QT</td>
<td>217</td>
</tr>
<tr>
<td>Cross-platform + Xamarin</td>
<td>28</td>
</tr>
<tr>
<td>Cross-platform + Phonegap</td>
<td>90</td>
</tr>
<tr>
<td>Health + care + Bluetooth</td>
<td>478</td>
</tr>
</tbody>
</table>

Table 3.1: Search results from KTH Primo [26-03-2015].

In the initial search, general terms were used to get an idea of the volume of available information. The first general term was ”cross-platform” which generated 11,253 results. Due to the volume of these results, relevant terms were added in addition to the first term to narrow down the results. This process was iterated in combination with different search terms to yield the most relevant sources. The number of results from each iteration is given in Table 3.1.

3.2 Working methodology

The method used for the practical part of the project was a variant of Scrum. This version of Scrum included the regular meetings every week with the stakeholder. During the meetings the progress from previous meeting was shown as a demo and could be discussed. After the demo new goals were set up for the next meeting. The goals were shaped by the stakeholder as to cater to their priorities.

The goals set by the stakeholder could then be divided into smaller, more technical tasks that the students focus on. What was not included in this version of Scrum was the roles seen in typical Scrum teams. This was not included because the project group only contained two people which made the roles superfluous as there are generally more people in a standard Scrum team.
Chapter 4

Cross-platform development

The idea behind cross-platform development is that the application can easily made compatible with other platforms. With a well defined structure, the transition to other platforms can be streamlined. The structure that was designed for this application is described in this section. A discussion on the chosen platform is also done.

Developing an application with the purpose of making it cross-platform comes with the burden of generality and with that certain costs must be considered. On the contrary cross-platform application have advantages which have to be weighed against the negative aspects.

4.1 Pros and Cons of cross-platform development

In this section the main positive and negative aspects are discussed as well as a summary of the conclusions made from the discussion.

Positive aspects:

- The service cost incorporated with the application becomes significantly less as deploying updates and patches becomes easier to implement. Assuming a proper structure, the cost of developing of extending the application across multiple platforms is decreased. The already established applications can serve as a guide for new platforms to follow. The documentation becomes easier to manage as the API’s can be held common for all platforms.

- From a users point of view the consistency between the platforms in terms of graphics and functions give a sense of comfort.

Negative aspects:

- Native features may not be available for cross-platform application. This can be due to the nature of the function of variable and how it interacts with the rest of the
program. As cross-platform applications work by connecting multiple programming languages, the medium for doing this may not always support the functions. This has mostly to do with the infancy of the technology [13].

- **Limitations in terms** of developing platforms. When programming general applications, the developer has to conform to the standard of the tools in use which are helpful but at the same time limiting. There is little flexibility available and this may cause frustration for the developer.

- **Inefficiency** in terms of code rendering time as the development tools may prove to be bottlenecks. As the functions calls are made across languages, the translation time has to be factored in and will be slower than a native function call.

Even though there are many negative aspects of cross-platform development, the positive aspects still outweigh the negative ones. This was because the whole idea with the project was to develop an application suited for the future market. Given the trends it is not feasible to develop for a single platform.

### 4.2 Cross-platform development tools

The deciding factor in choosing a development platform was the BLE compatibility as this was also the most constraining component to the project. Some other factors to consider were the available resources, familiarity with the language and performance. The chosen platform would also have to be relatively established as this project is designed for future development. A list of these criterion can be seen in Table 4.1 together with the contending platforms.

<table>
<thead>
<tr>
<th>Platform</th>
<th>Free BLE compatibility</th>
<th>Multi-platform support</th>
<th>Online Resources</th>
<th>Familiarity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Phonegap</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>x</td>
</tr>
<tr>
<td>Xamarin</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>x</td>
</tr>
<tr>
<td>Qt</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Unity</td>
<td>x</td>
<td>✓</td>
<td>✓</td>
<td>x</td>
</tr>
</tbody>
</table>

**Table 4.1: Criterion for the development platforms**

- QT was the option for the development of this application as it fulfilled majority of the criterion. The closest contender was Xamarin as it also fulfilled at least as many, if not more criterion. What put QT ahead was the familiarity with the languages which added a level of comfort.
• **Xamarin** was considered to be the main contender to QT as it fulfilled a lot of the criterion. The working language is C# which would have been an easy language to learn due to prior experience in C and Java. Xamarin also has cross-platform support for BLE which would have been advantageous in the context of this project. In addition to this, there are several examples demonstrating the different functions of BLE [5].

• **Phonegap** was another alternative, however this option had obvious drawbacks. Applications are written in HTML, Javascript or CSS and the application ends up being a hybrid application [1]. The Phonegap languages can be connected to the native APIs enabling BLE support. This does however have a layer of complexity to it and is not as simple as using predefined functions from a library.

• **Unity** is a popular development platform primarily focused around game development. What made it relevant for the context of this project was its vast platform support. Among all contenders, Unity had the largest range of support of platforms. A dissuading factor that put the other tools ahead of Unity was the cost. To get access to the BLE support, the plugin needed to be purchased for 150$ [25]. Given the fact that both Qt and Xamarin had this support for free, Unity was not chosen.

As the points above demonstrate, there are a lot of options for developing cross-platform applications and Qt was just one of the approaches.

4.3 Our choice in detail

QT is a powerful platform with an intuitive API and a lot of support on the internet from different forums. Due to this, it is easier to get help when problems present themselves. Another reason to why the platform was chosen, was because it supports Android BLE and also have plans to expand the support to other operating systems. If this support exists in the future, the native part of the application can more or less be taken away and have full cross-platform support as well. Taking out the native part would increase the productivity and decrease the cost of the application because more operating systems would be supported by the same code.

4.3.1 Programming languages in QT

The GUI is programmed in a language called QML which is a language developed by QT. QML is very similar to XML and HTML in the way that it is mostly used for graphical programming. The QML also has support for Javascript which lets the developer write simpler functions right in the graphical part. If the user wants too, the whole application can be written purely in QML, but is not very efficient if it has to make a lot of calculations and heavier jobs. The GUI is extended with C++ which lets the programmer make the calculations and heavier work.
4.3.2 The development environment

The environment in QT is pretty straight forward. Useful examples and recent projects are accessible directly from the start window. The environment has a built in debug system which unfortunately does not support debugging for Java. To enable Android development in QT, the correct libraries have to be installed in the right way. An extensive tutorial on how to do this can be found online [6]. Unfortunately the QT environment does not support live syntax correction for Java, but it does support error handling at compile time.

4.3.3 Software licensing

The version of Qt license that was used during the project was the free license too keep the development as cheap as possible. The free license of Qt goes under three different licenses [7], LGPLv2.1, LGPLv3 or GPL v3 and some modules goes under the LGPLv3 only. This means that the application developed in the environment can’t be used for profit as it would violate these licenses. To be able to sell the application developed in this thesis, the commercial version of the environment would have to be bought. Buying the commercial license of the Qt environment would give the developer more tools than in the free version which would aid the further development of the application.

4.4 Defining JNI

The cross-platform compatibility for this project was achieved through the use of JNI or Java Native Interface. JNI is a programming framework for making the Java interface available to other languages [11]. In this project, Java functions are called upon from C++ through JNI. In QT a lot of the low-level work is not visible to the programmer, however in the background a two-step referencing system is at work. JNI provides a pointer to a function table containing all the functions in the Java class. Each function in the table can be accessed by adding an offset to the table pointer. This allows the C++ environment to access the Java functions. Figure 4.1 depicts this in an intuitive manner.

![Figure 4.1: Design overview for JNI](image)

14
4.5 Cross-platform structure of the application

To aid the explanation of the structure, a depiction of the structure is shown below in Figure 4.2. The program can be seen as a three part structure where each part is discussed in detail below.

**Figure 4.2: Hierarchy of HealthyWay**

4.5.1 The native module

As well as being the module that communicates with the hardware, it also represents the non-static part of the code. In the HealthyWay project so far, the native code was written in Java which made it compatible with the Android operating system. The QT environment is not yet powerful enough to enable the developer to write Bluetooth code portable enough for the scope of this project. The involvement of BLE increased the complexity of the program by quite a lot due to the use of GATT profiles. The data that arrives from the hardware is handled by the native module primarily. To address the requirement of modularity, the native module was designed with the intention of easing the addition of new devices.
4.5.2 The logic module

This module can be seen as the bridge between the native module and the cross-platform support. QT makes use of C++ as the primary language together with custom made data types suited for the QT environment. This module was therefore defined in C++ which also eased the integration with the graphic environment. Another key component defined in this module are the header files containing the prototype declarations of the C++ functions as well as the signal and slots. For a developer looking to build upon this platform, the header files provide a great outline as to which functions and variables need to be defined. Since the GUI is already made to work with the C++ code, the graphical part can be left unchanged. This is described in greater detail in the next chapter.

4.5.3 The GUI module

From a system point of view the GUI module is the top layer that directly interacts with the user. QML was the language of choice here mostly because of its natural interaction with the QT environment. Each page was given its own file to make the program structure more intuitive. This also builds upon the modularity of the program as it makes it easier to create new pages if that would be required. As mentioned earlier the consistency in terms of looks is something that was seen as a strength and this module is what enables this. If a new platform was to be introduced, the GUI module could in theory be left untouched resulting in an identical application from a users point of view.
Chapter 5

The application in detail

In this chapter the application is discussed in detail. This includes a detailed description of all the modules together with instructions on how to exchange parts. One thing to note is that this project has been developed on a Sony Xperia Z1 [18]. The reason behind the choice of device was simply that the device was the only one the students could access during the project. Running this project on another Android device might give a different result because of different hardware, software and pixel density of the screen.

5.1 The cross-platform parts

In this section of the report a detailed explanation of the cross-platform parts in the application is explained. These parts of the project have been developed to be unchanged regardless of the operating system.

5.1.1 The GUI module

The GUI is purely written in QML and has direct communication with the logic module. The modules are connected via the main.cpp file which passes a pointer of the heatlhywayfunctions class to the QML. This way the QML-files have direct access to all the functions in that class. Another way these parts are connected are via signals and slots. The idea of signals and slots is that if a signal is sent from one side, say the logic side, a slot on the GUI side can be activated and vice-versa. The slot is the function that the signal should invoke. For example could a slot that calculates a value be invoked when the value is changed and the signal is sent. To connect to one of these signals the QML has a property called Connections. By connecting to signals from the logic module the QML can behave in different ways. In Listing 5.1 is an example of how signals are used to show different canvas areas depending on what characteristic has been chosen.
In the application the GUI is split into three pages:

- **main.qml**, opens when the application is started and shows a list of nearby devices when *Scan* is pressed.
- **servicePage.qml**, opened when the device is connected to another device and shows available services on the remote device.
- **dataPage.qml**, shows the data from a characteristic that was chosen on the previous page.

These three pages are connected via a *Loader* [20] that sets the current page with the *source* command. By using this it is very easy to extend the GUI with new pages and linking them using buttons or actions. When a device is chosen on the main page, the Loader loads the service page and requests a list of the available services on the remote device. The list is then shown in the service page and pressing one of the items in the list leads to the data page where data can be shown. To get a better understanding on how the whole application works from the GUI all the way down, look at Figure 5.1.

An interesting part of the GUI is the *canvasArea* located in the **dataPage.qml** file. This rectangle contains the graphics connected to the characteristic currently sending data. For example the ECG characteristic shows the readings as a graph and the battery level is shown as a bar. As mentioned earlier, the graphic will depend on which signal is caught from the *Connection* property ”showGraphics” in Listing 5.1. Everything inside the *onPaint* clause is shown on screen and has to be repainted every time the values of the characteristic is changed. Repainting is made through the ”requestPaint()” command which is called through another *Connection* that awaits new values for the canvas.

Another useful feature is the screen that pops up when the device is scanning for remote devices. This feature is practical as it visually alerts the user as the alternative is to simply lock the user interface. This was done by using *Connections* to show the scanning status while the device is scanning and at the same time disabling the other buttons. When the scanning is done, the window is hidden and the buttons are enabled again.
Figure 5.1: Flowchart of the application when pressing the "Scan" button

In Figure 5.2, three screenshots from the application showing the navigation through the application when going from the scan result to the ECG measurement page. Starting from the left is main.qml, servicePage.qml and dataPage.qml.
5.1.2 The logic module

The logic module is written in C++ and its purpose is to transfer data and functionality from the native code to the QML. This module makes its variables visible to the GUI through something called a \texttt{Q\_PROPERTY}. By using this the QML has direct access to that variable and is notified via signals every time the variable is changed. An example of a \texttt{Q\_PROPERTY} is the "deviceList" which holds the list of remote devices. The QML uses the "getDeviceList" to access the list of strings inside the "deviceList" when the signal "deviceListChanged" is emitted from the logic module.

For the QML to be able to use the functions of the C++ the property \texttt{Q\_INVOKABLE} is used. This gives the QML complete access to that particular function. The logic module has four main files:

- \texttt{main.cpp}, makes the functionality of the C++ available to the QML and initiates the application. This includes initiating the \texttt{view} which sets the \texttt{main.qml} as the starting window. In the \texttt{main} class the connection between the GUI module and the logic module is created. This is done by creating a pointer to an object of the \texttt{healthywayfunctions} class and passing it to the QML.

- \texttt{healthywayfunctions.cpp}, contains all the functions that the QML uses and processes data from the native code. This class uses the information it gets from the \texttt{jnihealthyway} class and converts it so that the QML is able to display it. For example the ECG data needs to be separated into individual samples and be given a time stamp so the graph can display the information appropriately.
• **mythread.cpp**, creates a thread that calls a specific function in a loop until it is stopped. It is a primitive class sends a signal on a specified interval to the `healthywayfunctions` class which activates the appropriate functions. The threads are used mainly for polling information from the native code because the native module can’t call the logic module directly.

• **jnihealthyway.cpp**, communicates directly with the native code, processes the data and sends it back to the caller. The main idea behind the class is to bridge the gap between the native module and the logic module. This class is specific to the native code and is not compatible with all operating systems, although the structure of the class should be the same regardless the system.

Because BLE devices return different data types depending on the characteristic chosen, the logic module must know about the different characteristics. With this in mind there are separate functions in the `jnihealthyway` class which are adapted to the specific characteristic. Starting from the top, the first encounter of the specific characteristic code is in the function `updateData`. The outline of the function can be found in Listing 5.2.

```cpp
void HealthyWayFunctions::updateData()
{
    if(chosenCharacteristic == "Battery level indicator") {
        // Code to handle battery level information
    } else if(chosenCharacteristic == "Manufacturer name") {
        // Code to handle manufacturer name information
    } else if(chosenCharacteristic == "ECG measurement") {
        // Code to handle ECG data
    } else {
        qDebug() << "CHARACTERISTIC NOT FOUND";
    }
}
```

**Listing 5.2: Example of updateData from healthywayfunctions.cpp**

Each clause in Listing 5.2 calls a corresponding function in the `jnihealthyway` class. The functions in that class are in turn responsible for calling the appropriate method/function in the native code.

Communication between different programming languages is not always an easy task. The QML and C++ work together in an intuitive way as they have been integrated with each other through QT. The communication between C++ and Java is however a lot trickier. As mentioned earlier, C++ is able to directly call Java methods through libraries in QT, but there is little to no support for communication from Java to C++. There are ways of doing this but due to the time constraint this was not implemented. Instead, polling was made use of because of the simple implementation. The simplicity did however, come with the drawback of poor performance and little reliability.

The polling starts with the user pressing the "Scan" button (as shown in Figure 5.1) or one of the characteristics. When pressing for example the "Scan" button, `healthywayfunctions`
starts a thread by calling the `mythread` class which in turn triggers a loop. For each loop, `mythread` sends a signal to `healthywayfunctions` which invokes the code in Listing 5.2. The loop stops if the `Stop` variable for that thread is set to `true`. Delays of the loop is set with the `sleepTimer` variable. This is useful as different characteristics need different sleep values, for example the battery level does not need one update every 40 milliseconds while the ECG data needs it.

5.2 The native part

This part of the application is made to be specific for one operating system and should syntactically look different for each operating system.

5.2.1 Native Android

In QT there are libraries that support communication between Java and C++ called QAndroidJniObject [22] and QAndroidJniEnvironment [21]. A graphical representation of this can be found in Figure 4.1. By making use of these libraries the C++ side of the connection can invoke Java methods. Here in Listing 5.3 there is a simple example on how QAndroidJniObject can be used to call Java methods. To see more complex examples see the application documentation or the API from QT [23].

```
void JNIHealthyWay::turnBluetoothOn()
{
    QAndroidJniObject::callStaticMethod<void>("org/qtproject/example/notification/NotificationClient","btON");
}
```

Listing 5.3: Native C++ from jnihealthyway.cpp

The QAndroidJniEnvironment is mainly used to convert Java data types into C++ code which is returned to the main application.

The Java file is the largest file as it contains the necessary bits of code to correctly handle the communication between the hardware and the application. Both the scanning action and the fetching of characteristic data is handled with callbacks. These callbacks are what listens for changes and react when information is received.

There are three main characteristics implemented. These characteristics are the battery level, the manufacturer name and the ECG reading. Both the battery level and the manufacturer name use the basic "readCharacteristic" functionality that reads the value of a variable containing the specific data on the remote device. The ECG data on the other hand uses the `notify` feature which sends data continuously until the notification is stopped. The process of using the ECG measurement characteristic starts by writing the `enable notification value` to the descriptor of that characteristic. When the descriptor has been written the application
writes a byte array containing "01" to tell the remote device to start reading the ECG data using its sensors. When this is done the native code receives a steady stream of data from the characteristic until a byte array containing "11" is written to the device. The ECG data is a 20-bit long array containing the time stamp followed by 6 sample readings. These values are split into separate integers in an array, which are then put in a queue to wait for the main application to process.

5.3 Installing new modules

Because the structure of the application is modular, parts of the system can be changed without having to write a whole new application. In this section the information on how these modules can be exchanged or built upon.

5.3.1 Steps for supporting other operating systems

The first step is to copy the content of the jnihealthyway.h file. This file contains the structure and definitions that is needed to be compatible with the logic module. The file cannot be used directly with its jnihealthyway.cpp because it is specific for the Android operating system. In the functions that are defined in the header the programmer has to do the necessary calls to the native module and conversions from native variables to C++ variables.

The biggest part of integrating a new operating system is to write the native code for that specific system. It is in the native code that all the Bluetooth communication takes place and how to handle characteristics. Because the native part of the program could be written in a variety of different languages intended for different operating systems, the native code will differ from one another. With that said, the Java file NotificationClient.java serves as a guideline to understand what basic functionality is needed of the native code.

5.3.2 Steps for adding new characteristics

There are mainly two steps that has to be done to add a characteristic to the application. Figure 5.3 shows the basics of how to add a new characteristic to the application.

As shown in Figure 5.3 the yellow part is what makes up the cross-platform part. A change in this segment will alter the logic for all operating systems while the red part only applies to the Android native code. In the logic module the program needs a Q_PROPERTY in order to make values of the new characteristic available for the QML. An example of this is the Q_PROPERTY "batteryLevel" which sends the battery level in percentage to the QML so it can draw the bar showing it.
Figure 5.3: A basic picture of what has to be done to add a characteristic

5.3.3 Documentation

A well documented code was also the result of the project where variables, functions and classes are explained as detailed as possible. This was of high importance as to ease the continuation of the project so that anyone with a similar area of expertise could continue the development. The documentation was written mostly in a doxygen standard to create an easy to follow text without the clutter of big code chunks. Doxygen generates a online-documentation with navigation between the different files and functions which is more pleasing than reading comments in the actual code. This method worked with the C++ and Java code as they had built in support for this. Doxygen does however not support QML and had to be documented only in the .qml files.

5.3.4 Version control

For version control Git was made us of where the repository was provided by Github. GitHub has many different features but one of the major ones is that the user can go back to previous versions of the application. Another feature is the ability to merge different code blocks into a single code block. This feature was used when the students worked on two different parts. With the merge ability the students could combine both blocks into the main application.
Chapter 6

Conclusion

While the goals for this project were successfully achieved there are many areas which could be improved. The project was meant to be a platform to build upon and in many ways is meant to be a futuristic project rather than something that is ready for deployment in its current state. The most important aspect this project has succeeded with is the ability to successfully synergize three different programming languages into a functioning application. While this aspect provides enormous customizing capabilities, it may also prove to be the demise of the application.

On further reading about the different development platforms we saw that Xamarin might have been a better alternative than QT. This is because we found a couple of examples of BLE applications which both iOS and Android could run.

As can be seen in Figure 5.2 the GUI is quite basic while at the same time intuitive. Because the GUI had such a low priority, the back-end of the application was made in a much more detailed manner. The application is far from perfect but serves as a good framework and prototype for the project of building a commercial product.

6.1 Reflecting over the goals

The goals that were stated in the introduction can now be alluded to as the project has been carried out.

- **Goal 1:** This goal has been successfully achieved as we have learned a lot about BLE and wearable electronics.

- **Goal 2:** We strongly believe that the choices that have been made in terms of development platform and programming languages will make this application scalable. The fact that there is a clear distinction between the native and non-native part easily enables users to make their own platform compatible with the HealthyWay application.
• **Goal 3**: This goal has been achieved through proper planning prior to the development process. By working towards a modular structure to the program, new services can easily be integrated into the program. We stress future developers to continue this modular structure when developing integration with new platforms.

• **Goal 4**: With the assumption that future developers have a decent background in programming, the provided documentation should serve as a reference. The use of doxygen is to aid future students due to the intuitive style in which the documentation is created.

The goals have been achieved to large extent however, a lacking aspect is the testing as to see what effect the application has on the lifestyle. From a technical point of view the application has a structure that lives up to the aforementioned goals.

### 6.2 Known problems

One problem with the interface is that if the label called *Data:* seen in the last picture of Figure 5.2 is pressed, the application reverts back to the *servicePage*. The other problem with the application is that there is no connection status shown to the user when choosing a device. Because the GUI doesn’t respond right away the user could get the feeling that nothing is happening which could be misleading. To solve this the connection should be put in a separate thread just like the scanning procedure.

### 6.3 Future improvements

Before explaining in depth what could be specifically improved it is important to state that given the time frame and the infancy of the tools used were at, the solutions may have been suboptimal. With improvements to the QT environment, the application could be made a lot more efficient. A major bottleneck in terms of performance was the use of polling. Given the rate at which the ECG data was sent, the cross-platform calls had to be done frequently making the ECG function very expensive. On top of that, the data was to be visualized through a graph which in itself drew a lot of resources. The way the application collects the data in itself is a very heavy process and is what should be improved. In its current state, the communication between the languages can only be initiated from the C++ side. This is the reason why polling had to be used. In an ideal case the data should be forwarded to the GUI directly from the native interface. If this was made possible, the use of polling would be obsolete and performance would greatly improve. This also ties into the next point which is expanded upon below.
A feature that would be of great value is to be able to connect to multiple devices at the same time instead of going back each time the user wants to change device or characteristic. A checklist would be a great addition to the application where the user can choose multiple devices and characteristics and see the data from the devices in one continuous page or multiple windows to swipe through. If this application is built upon and scaled up to include several devices, the polling solution will not be viable. This is a key point as it directly relates to one of our main goals for this project. Structurally, the application can add compatibility, however from a practical point of view it will be a very inefficient solution and not one that would be ready for deployment. The very first improvement a future developer would therefore have to make would be to find a way to bypass the polling solution and achieve the same result. By doing this the application is scalable even on a practical level.

At the moment of writing the only characteristics we had time to implement were for battery level, manufacturer name and ECG samples. Increasing the number of characteristics would increase the usefulness of the application a lot. This can be done by following the instructions in chapter 5 or by reading the documentation of the application.

With the increasing popularity of wearable electronics, a realistic idea would be to extend the application to smart watches. Notifications to remind the user about medication, battery levels and anomalies are just a few of the many possible ideas. As the framework is made to integrate these kind of ideas we believe that HealthyWay is a product made for the future.

6.4 The future of QT and HealthyWay

The timing for the development of this application is rather unfortunate as a new and very relevant update for the QT environment is on the verge [13]. QT version 5.5 will have support for BLE across both Android and iOS. The extent of the BLE support is yet to be tested, however in an ideal case, the native code can be made obsolete.

A crucial update that needs to be addressed is the visualization of the incoming data which has been partially neglected in this project. The reason behind this are the priorities that were made by the stakeholder as well as the lack of resources. In the free version of QT the graphical capabilities are limited which made the data visualization a very challenging segment.
6.5 Other areas of application

HealthyWay is not restricted to being a health-care application. The code has been written in such a way that it can be adapted to any form of characteristics and also its functionality. Just because health-care was used as a focus point in this project does not mean that is the only possible use of the application. As an example the application could be adapted to serve as an environmental analysis device. By adding the necessary characteristics (such as wind speed or barometric pressure) and, if wanted, a range of graphical features the application could tell the status of the environment instead of the health of the user. There is no limitation to what the application can be used for and in what combination all of these characteristics can be used, only the ingenuity of the programmers and hardware developers can set the limitations.
Bibliography


[8] IRVING ROTHCHILD. INDUCTION, DEDUCTION, AND THE SCIENTIFIC METHOD. Technical report, Case Western Reserve University School of Medicine, Cleveland, Ohio, 2006.


Appendix A

HEALTHY WAY, Bluetooth LE Protocol Specification V2.0
**GATT Settings Specifications:**

**Health Device:**

The Health Device has the following roles,

<table>
<thead>
<tr>
<th>Services</th>
<th>Role Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>ECG Service</td>
<td>M</td>
</tr>
<tr>
<td>Respiration Service</td>
<td>M</td>
</tr>
<tr>
<td>Acc &amp; Gyro Service</td>
<td>O</td>
</tr>
<tr>
<td>Device Information Service</td>
<td>M</td>
</tr>
<tr>
<td>Battery Service</td>
<td>M</td>
</tr>
</tbody>
</table>

M - Mandatory  O - Optional

**ECG Service:**

This service is mandatory. Our Health device is always equipped with the ECG sensor. This service is used in Advertisement packet.

**Base UUID is --**  4E1B****-81AA-11E2-9E96-0800200C9A66

<table>
<thead>
<tr>
<th>Service/Characteristic</th>
<th>UUID</th>
<th>Custom/BLE SIG</th>
</tr>
</thead>
<tbody>
<tr>
<td>ECG Service</td>
<td>4E1B94B0-81AA-11E2-9E96-0800200C9A66</td>
<td>Custom</td>
</tr>
<tr>
<td>ECG Measurement Characteristic</td>
<td>4E1B94B1-81AA-11E2-9E96-0800200C9A66</td>
<td>Custom</td>
</tr>
</tbody>
</table>
### ECG Service Characteristic

<table>
<thead>
<tr>
<th>Characteristic</th>
<th>Role</th>
<th>Value Type</th>
<th>Property</th>
</tr>
</thead>
</table>
| ECG measurement characteristic | M    | ECG Data: Uint_24  
Time stamp: Uint_16 | Notify |
| ECG Sampling Rate & Reference Voltage Characteristic | M    | Sampling rate: Uint_16  
Reference Voltage: Uint_8 | Read |
| ECG Write Characteristic | M    | Uint8 | Write without Response |

**M - Mandatory**

### ECG Measurement Characteristic:

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Payload Length</th>
<th>Value Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time stamp</td>
<td>2 bytes</td>
<td>0 - 1023</td>
</tr>
<tr>
<td>ECG Data 1st set</td>
<td>3 bytes</td>
<td>0 to 16,777,215 (ADC output)</td>
</tr>
<tr>
<td>ECG Data 2nd set</td>
<td>3 bytes</td>
<td>0 to 16,777,215 (ADC output)</td>
</tr>
<tr>
<td>ECG Data 3rd set</td>
<td>3 bytes</td>
<td>0 to 16,777,215 (ADC output)</td>
</tr>
<tr>
<td>ECG Data 4th set</td>
<td>3 bytes</td>
<td>0 to 16,777,215 (ADC output)</td>
</tr>
<tr>
<td>ECG Data 5th Set</td>
<td>3 bytes</td>
<td>0 to 16,777,215 (ADC output)</td>
</tr>
<tr>
<td>ECG Data 6th Set</td>
<td>3 bytes</td>
<td>0 to 16,777,215 (ADC output)</td>
</tr>
</tbody>
</table>

Allowed payload length is 20 bytes. So from above table we have reached the limit.

**Time Stamp:**

Time stamp is not the real time. But, it is just for the data order or to track the missing data. The value is from 0 - 1023. It works as a count so once reaching 1023 the value again starts from 0.

**ECG Data:**

We are about to use the 24 bit resolution ADC for processing ECG data. So each ECG data is of 3 bytes.

**Byte order in Payload:**
**ECG Write Characteristic:**

This characteristic is used to let know the peripheral device that Mobile needs ECG service i.e. ECG measurement Data. So, mobile after connection can send value 01 to peripheral device in order to get ECG data.

If user want to stop receiving ECG data then we can use '11' to stop.

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Payload Length</th>
<th>Value Range</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>ECG Write Char</td>
<td>1 Byte</td>
<td>0 - 255</td>
<td>01 or 11</td>
</tr>
</tbody>
</table>

**ECG Sampling Rate & Reference Voltage Characteristic:**

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Payload Length</th>
<th>Value Range</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sampling Rate</td>
<td>2 Bytes</td>
<td>0 - 1023</td>
<td>250</td>
</tr>
<tr>
<td>Reference Voltage</td>
<td>1 byte</td>
<td>0 - 255</td>
<td>33 (3.3 V) or 50 (5V)</td>
</tr>
</tbody>
</table>

**Sampling Rate:**

For ECG sampling rate between 200 - 500 Hz were recommended to be used as per Medical body. we are about to use the **250 Hz** Sampling Rate i.e the ADC sample the signal at the rate of 250 Hz. With the increase in sampling rate the data rate increases and data can get lost.

The Sampling rate is needed to be sent to mobile device since we can use them to sample the received signal.

**Reference Voltage:**

The Reference voltage can be of 3.3 v or 5 v.

The mobile should know the used Reference voltage in order to process the received ECG data which will be in order of bytes.

**Note:** If we use 3.3 V then Peripheral Device will send 33 and in mobile we divide them by 10. Same as for 5 V
Byte order in Payload:

<table>
<thead>
<tr>
<th>Sampling Rate MSB</th>
<th>Sampling Rate LSB</th>
<th>Reference Voltage</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 Byte</td>
<td>1 Byte</td>
<td></td>
</tr>
<tr>
<td></td>
<td>2 Bytes</td>
<td></td>
</tr>
</tbody>
</table>

**Respiration Service:**

This service is mandatory. Our Health device is always equipped with the Respiration sensor

**Base UUID is --** 4E1B****-81AA-11E2-9E96-0800200C9A66

<table>
<thead>
<tr>
<th>Service/Characteristic</th>
<th>UUID</th>
<th>Custom/BLE SIG</th>
</tr>
</thead>
<tbody>
<tr>
<td>Respiration Service</td>
<td>4E1B94C0-81AA-11E2-9E96-0800200C9A66</td>
<td>Custom</td>
</tr>
<tr>
<td>Respiration Measurement Characteristic</td>
<td>4E1B94C1-81AA-11E2-9E96-0800200C9A66</td>
<td>Custom</td>
</tr>
<tr>
<td>Resp. Sampling Rate &amp; Reference Voltage Characteristic</td>
<td>4E1B94C2-81AA-11E2-9E96-0800200C9A66</td>
<td>Custom</td>
</tr>
<tr>
<td>Resp. Write Characteristic</td>
<td>4E1B94C3-81AA-11E2-9E96-0800200C9A66</td>
<td>Custom</td>
</tr>
</tbody>
</table>

**Respiration Service Characteristic**

<table>
<thead>
<tr>
<th>Role</th>
<th>Value Type</th>
<th>Property</th>
</tr>
</thead>
<tbody>
<tr>
<td>M</td>
<td>Resp. Data: Uint_24</td>
<td>Notify</td>
</tr>
<tr>
<td></td>
<td>Time stamp: Uint_16</td>
<td></td>
</tr>
<tr>
<td>M</td>
<td>Sampling rate: Uint_16</td>
<td>Read</td>
</tr>
<tr>
<td></td>
<td>Reference Voltage: Uint_8</td>
<td></td>
</tr>
<tr>
<td>M</td>
<td>Uint8</td>
<td>Write Without Response</td>
</tr>
</tbody>
</table>

M - Mandatory
Note: If we use 3.3 V then Peripheral Device will send 33 and in mobile we divide them by 10. Same as for 5 V

**Byte order in Payload:**

<table>
<thead>
<tr>
<th>Sampling Rate MSB</th>
<th>Sampling Rate LSB</th>
<th>Reference Voltage</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 Byte</td>
<td>2 Bytes</td>
<td>1 Byte</td>
</tr>
</tbody>
</table>

**Device Information Service:**

The above service is useful to uniquely identify each device. Since IOS doesn't allow to read the MAC address of device it would be very difficult to differentiate devices with same service and in this case the device information service can used.

Usually, **device Name** used in general to **Identify** the devices. We can use either Full name or shortened name. Device Name is Shortened (String length) in order to accomodate it with the advertisement packet. The device name, transmitt power level etc are usually sent in scan response payload.

*For eg:* If we set Device Name as 'BT_Device_Name' then the shortened name is 'BT_Dev'
  'Medical_Box' --- 'Med_Box'
  'ECG_Dev' etc.

User can opt for either ECG_DEV (or) Med_Box depending upon the need.

We can change the device name thro' air (mobile) after connection. So when user buys the product first they can initialize connection and after connection they can change the name of the device as they wish with some limitations in size of characters. So the user can easily identify the device next time.

To set the name, We should however use the `sd_ble_gap_device_name_set()` method instead of doing it directly with the handle, as that will be portable to all future sofdevice version as well. The handle may however be useful when receiving a write to this characteristic. I.e through mobile phone we can write the device name using Write Characteristic.

Also, if we are advertising while changing the name, we have to make sure to set up the advertising data again, essentially calling `ble_advdata_set()` again, after having changed the name, since this can't be done automatically. I.e just to turn the device off and on to see the changes.
Along with the device ID we can include the security and bonding procedures for authentication. Bonded connections are in a way "paired" to the iOS device. Once bonded they will not communicate to any other device. This ensures the device will always have a secured connection with the bonded iOS device.

Usually, the device MAC address can be used to identify each device but IOS has restricted this due to security reasons. IOS uses private address to identify the device and this keeps on changing from time to time so it's difficult to keep an track on the device.

**Note:**
*Further Device Information, Such as Manufacture name, model number, software rev, Frimware rev etc can be retrieved by setting the secondary service. The above info is sent to mobile after connection establishment between Harware_Unit and Mobile_Device through secondary service & its characteristic.*

<table>
<thead>
<tr>
<th>Service/Characteristic</th>
<th>UUID</th>
<th>Custom/BLE SIG</th>
</tr>
</thead>
<tbody>
<tr>
<td>Device info Service</td>
<td>0x180A</td>
<td>BLE SIG</td>
</tr>
<tr>
<td>Manufacturer name string Characteristic</td>
<td>0x2A29</td>
<td>BLE SIG</td>
</tr>
<tr>
<td>System ID Characteristic</td>
<td>0x2A23</td>
<td>BLE SIG</td>
</tr>
<tr>
<td>Software Revision String Characteristic</td>
<td>0x2A28</td>
<td>BLE SIG</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Device info. Service Characteristic</th>
<th>Role</th>
<th>Value Type</th>
<th>Property</th>
</tr>
</thead>
<tbody>
<tr>
<td>Manufacturer name string Characteristic</td>
<td>M</td>
<td>Utf8s</td>
<td>Read</td>
</tr>
<tr>
<td>System ID Characteristic</td>
<td>M</td>
<td>Manufacturer Identifier: unit40</td>
<td>Read</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Organizationally Unique Identifier: unit_24</td>
<td></td>
</tr>
<tr>
<td>Software Revision String Characteristic</td>
<td>M</td>
<td>Utf8s</td>
<td>Read</td>
</tr>
</tbody>
</table>

M - Mandatory

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Payload Length</th>
<th>Value Range</th>
<th>Value</th>
<th>Length for the value we used</th>
</tr>
</thead>
<tbody>
<tr>
<td>Characteristic</td>
<td>Description</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Manufacture name String</td>
<td>It's the string format describing the name of the manufacturer. The length depends upon the number of characters used to describe the manufacturer name. Max of 20 bytes allowed.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Byte Order:</td>
<td>H</td>
<td>T</td>
<td>K</td>
<td>_</td>
</tr>
<tr>
<td>Note:</td>
<td>Everything should be represented in bytes. (Ascii value of corresponding bytes)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Software Revision String Character</td>
<td>It's important to keep track of software version. It's also in the string format. The length depends upon the number of characters used to describe the S/W version. Max of 20 bytes allowed.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Byte Order:</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Note:</td>
<td>Everything should be represented in bytes. (Ascii value of corresponding bytes)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>System ID Characteristic</td>
<td>The SYSTEM ID characteristic consists of a structure with two fields. The first field are the LSOs and the second field contains the MSOs. This is a 64-bit structure which consists of a 40-bit manufacturer-defined identifier concatenated with a 24 bit unique Organizationally Unique Identifier (OUI). The OUI is issued by the IEEE Registration Authority (<a href="http://standards.ieee.org/regauth/index.html">http://standards.ieee.org/regauth/index.html</a>) and is required to be used in accordance with IEEE Standard 802-2001.6 while the least significant 40 bits are manufacturer defined.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>If System ID generated based on a Bluetooth Device Address, it is required to be done as follows. System ID and the Bluetooth Device Address have a very similar structure: a Bluetooth</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Device Address is 48 bits in length and consists of a 24 bit Company Assigned Identifier (manufacturer defined identifier) concatenated with a 24 bit Company Identifier (OUI). In order to encapsulate a Bluetooth Device Address as System ID, the Company Identifier is concatenated with 0xFFFE followed by the Company Assigned Identifier of the Bluetooth Address. For more guidelines related to EUI-64, refer to http://standards.ieee.org/develop/regauth/tut/eui64.pdf.

Ref: Bluetooth Developer portal

Example:

If the system ID is based of a Bluetooth Device Address with a Company Identifier (OUI) is 0x123456 and the Company Assigned Identifier is 0x9ABCDE, then the System Identifier is required to be 0x123456FFFFE9ABCDE.

Byte Order:

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>..........</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
</tr>
</thead>
</table>

8 Bytes

Note: Everything should be represented in bytes. (Ascii value of corresponding bytes)

Battery Service:

It's important to know with Health Device Battery information.

<table>
<thead>
<tr>
<th>Service/Characteristic</th>
<th>UUID</th>
<th>Custom/BLE SIG</th>
</tr>
</thead>
<tbody>
<tr>
<td>Battery Service</td>
<td>0x180F</td>
<td>BLE SIG</td>
</tr>
<tr>
<td>Battery Level Characteristic</td>
<td>0x2A19</td>
<td>BLE SIG</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Device info. Service Characteristic</th>
<th>Role</th>
<th>Value Type</th>
<th>Value</th>
<th>Property</th>
</tr>
</thead>
<tbody>
<tr>
<td>Battery Level Characteristic</td>
<td>M</td>
<td>Uint8</td>
<td>0-100</td>
<td>Read</td>
</tr>
</tbody>
</table>

Mechanism of Health Device:

As of now, we have described the GATT profile for health device. Here we are about to explain the order in which packets (data) must be sent from peripheral device to mobile phone.