

Bachelorarbeit

cand. mach. Eduardo Jiménez Torres

Matrikelnr.: 2078404

Development of a bluetooth application for measuring tension forces of a turning jaw

wbk

Institute of Production Science
Karlsruhe Institute of Technology (KIT)
Kaiserstraße 12
76131 Karlsruhe

Prof. Dr.-Ing. Jürgen Fleischer
Prof. Dr.-Ing. Gisela Lanza
Prof. Dr.-Ing. habil. Volker Schulze

wbk

Institute of Production Science
Karlsruhe Institute of Technology (KIT)
Kaiserstraße 12
76131 Karlsruhe
Prof. Dr.-Ing. Jürgen Fleischer
Prof. Dr.-Ing. Gisela Lanza
Prof. Dr.-Ing. habil. Volker Schulze



Bachelor Thesis

for Mr. cand. mach. (B.Sc.) Eduardo Jimenez Torres, Matriculation Number 2078404,
Nancystraße 20, 76187 Karlsruhe

Development of a bluetooth application for measuring tension forces of a turning jaw

Entwicklung einer Bluetooth-Applikation zur Messung der Spannkraft an einer Spannbacke für Drehfutter

In clamping round workpieces for turning operations with lathe chucks there is no possibility to measure directly the clamping forces applied to the workpiece. Thin-walled work pieces incur deformations because of too high forces, while too low forces can cause a dangerous drop out of the work piece. Therefore a turning jaw measuring the clamping force is in development. To easily read out the sent data with smart phones or tablets, a bluetooth interface is implemented.

In this thesis a bluetooth application for measuring tension forces of a turning jaw has to be developed. Therefore the following tasks have to be accomplished:

- Description of all relevant bluetooth technology issues
- State of the art search for bluetooth and other radiocommunication applications in machine components, especially clamping devices
- Evaluation and selection of available transmission hardware, software version and programming tools
- Coding and implementation of an app to evaluate and display received clamping force data
- Validation of the functionality with provided sample data
- Characterization of perturbations in an industrial environment.

Internal thesis-number: MAP 0516
Date of issue: 2017-02-01
Date of submission: 2017-04-28
Guidance: M.Sc. Bastian Rothaupt

Karlsruhe, 2017-02-01

A handwritten signature in black ink, appearing to read 'Fleischer'.

Prof. Dr.-Ing. Jürgen Fleischer



Statement of Originality

I sincerely affirm to have composed this thesis work autonomously, to have indicated completely and accurately all aids and sources used and to have marked anything taken from other works, with or without changes. Furthermore, I affirm to have observed the constitution of the KIT for the safeguarding of good scientific practice, as amended.

Karlsruhe, May 8th 2017

Eduardo Jiménez Torres

Acknowledgement

First of all, I would like to extend my sincere gratitude to my supervisor Bastian Rothaupt. He has been helpful as I went through my thesis as we had weekly appointments so we could discuss about the thesis contents and talk about what I was doing.

He had adaptability as I am Erasmus student so the paperwork was more complicated to do and with the dates and times also.

As I was doing my last year of degree at the same time, he understood at every moment that I was busier than other normal student with the other subjects and comprehending that facts.

Finally, I would like to thank to all WBK members that helped me with this thesis.

Abstract

Bluetooth is a very common technology in our daily life and in industrial applications. In this thesis, bluetooth technology fundamentals, how it works and different types are explained. Bluetooth Low Energy, the main technology used is also explained deeply. How to choose a bluetooth technology, a bluetooth module and how to interpret the different specifications present in commercial and industrial datasheets, focus on a target group and focus the application development for them.

Within the development, how to structure the application and programming to achieve our objectives (evaluate and display received clamping force data) using own code and fitting java libraries to use them on the way we need to. Programming bluetooth code for an industrial application and validation of the functionality with provided sample data. Taking care of possible perturbations in an industrial environment.

In conclusion, have an overview of the whole process, from the first idea, through the investigation and comparison of technologies and devices to develop the application choosing IDE and OS. Programming knowledge is compulsory to understand the main body of the project.

Table of Contents

Table of Abbreviations	III
1 Introduction	1
1.1 Motivation	1
1.2 Objective	1
1.3 Structure of the Thesis	1
2 Background	3
2.1 Bluetooth Technology	3
2.1.1 What is Bluetooth?	3
2.1.2 Bluetooth class and version	5
2.1.3 Protocol and connection	8
2.2 Low Energy Bluetooth	10
2.2.1 Introduction to BLE	10
2.2.2 Key Terms and Concepts	11
2.2.3 GATT operations	13
2.3 Integrated Development Environment	14
2.3.1 Introduction to IDE	14
2.3.2 Android Studio	14
2.3.3 Java Language	16
2.3.4 XML Language in Android	18
3 State of the art	21
3.1 Bluetooth solution	21
3.2 Wi-Fi & ZigBee solutions	23
3.3 Summary	28
4 Own approach	30
4.1 Bluetooth module selection	30
4.1.1 HC-05	31
4.1.2 HM-10	33
4.1.3 Conclusions	34
4.2 Operating system & IDE selection	35
4.2.1 Operating system selection	35
4.2.2 IDE selection	37
4.3 Application development	38

4.3.1	Introduction to WBKAPP	38
4.3.2	Function flow in the application	40
4.3.3	Main classes and methods	41
4.3.4	Arduino code	49
5	Results	51
5.1	Validation of the functionality	51
5.2	User's guide	54
6	Assessment	57
7	Summary & Outlook	58
7.1	Summary	58
7.2	Outlook	58
	List of Figures	I
	List of Tables	II
	References	III
	Appendix	V
	ANNEX A	V
	ANNEX B	VIII
	ANNEX C	XIII
	ANNEX D	XVII
	ANNEX E	XIX
	ANNEX F	XXI
	ANNEX G	XXIII
	ANNEX H	XXV
	ANNEX I	XXVI
	ANNEX J	XXIX
	ANNEX K	XXXIV
	ANNEX L	XXXV
	ANNEX M	XXXVI
	ANNEX N	XXXVI
	ANNEX O	XXXVII
	ANNEX P	XXXVII
	ANNEX Q	XLII
	ANNEX R	XLII

Table of Abbreviations

Symbol	Measurement	unity
Vo	Bridge output voltage	[V]
V	Excitation voltage	[V]
Fg	Gage factor	[-]
E	Material module	[N/m ²]
Do	Tool diameter	[m]
Io	Area moment of inertia	[m ⁴]
T	Torque momentum	[Nm]
IDE	Integrated Development Environment	
IoT	Internet of Things	
BLE	Bluetooth Low Energy	
SIG	Bluetooth Special Interest Group	
IEEE	Institute of Electrical and Electronics Engineers	
AFH	Adaptive Frequency-Hopping	
F	Clamping force	[N]
Fm	Modified clamping force	[N]
D	Diameter	
K	Conversion factor	[mV/V]
Cm	Modified conversion factor units	[N/kN]
S	Error factor	[-]

1 Introduction

1.1 Motivation

“We must deal quickly with the fusion of the online world and the world of industrial production. In Germany, we call it Industrie 4.0.” (Angela Merkel, 2012)

Industry 4.0 using internet of things (IoT) is the future in factories as it helps to improve the production and to collect data so, for large enterprises and factories, it is an easy way to collect big data to improve production and so the efficiency. The basic principle of Industry 4.0 is that by connecting machines, work pieces and systems, businesses are creating intelligent networks along the entire value chain that can control each other autonomously. Over the last two decades, with the advance of ICT and web-based technology, a group of engineers and academic scientists have developed a concept of integrating all levels of the information systems in the industry, as presented at the National Academic Science and Engineering Conference at Germany. This concept is working with Internet of Things (IoT). Working on cyber-physical systems (CPS) and aiming for further smart manufacturing is seen as the beginning of Industrial 4.0; the new industrial revolution is leading the world into another stage in humans' industrial manufacturing history (Kagermann et al., 2014).

The development of a bluetooth application for measuring tension forces of a turning jaw is an easy way to supervise that a machine is doing its work properly as we can read the lectures of the sensors inside the machine as fast as possible using our Android powered device (either smartphone or tablet).

1.2 Objective

In the present work, it comes the development of an Android application having an overall vision of the process, from the first raw concept about discus about which platform is better to implement our application and choosing a bluetooth module technology according to the specifications we have to cover to the explanation of the whole java code that make the android application run as it does.

The application has a read function from the sensors that are implemented in the milling machine that are sent through a bluetooth module so they can be shown in our application and so, read by the worker that is controlling the proper functionality of each machine doing its work in real time way.

What is expected from this thesis is showing that internet of things (IoT) can be implemented in manufacture industry and that is helpful to control the machines and the industrial process (avoiding manufacture defects that can be detected by a peak in the force received by force sensors in the turning jaw e.g.).

1.3 Structure of the Thesis

Approach to achieve the objective the path we are going to follow is having an overall vision of the process. In chapter 2, we are discuss the background of the thesis, facing the different bluetooth versions and technologies and defining the characteristics that define each

one; in addition, we are going to discuss the different platforms we could develop our application and the languages used.

In chapter 3, we talk about the state of the art, in other words, what is already done about this field in manufacture industry, solutions that are already done, technology it is used but similar objectives as our applications (e.g. Wi-Fi, ZigBee), how they achieve the objectives and their deficits and pros from each industrial solution.

According to chapter 4, it is discussed the methodology we are going to follow so we achieve the objective, choosing an IDE (Integrated Development Environment) and a Bluetooth module from the different possibilities. Furthermore, we talk about the different logical functions that are implemented, the different code parts that make the application run and achieve the different functions that are implemented in it, shown by code extracts and looking for what we have to achieve.

In chapter 5, we talk about the results we have achieved during our thesis: a demonstration with some data collected from the force sensor and how to install the application in any device.

In chapter 6, we make an own critic about what we made and what we achieved, and showing the problems we had during this thesis' realization.

In chapter 7, a summary from the whole thesis and an outlook (future improvements, future implementation in manufactory industry) are done.

2 Background

2.1 Bluetooth Technology

2.1.1 What is Bluetooth?

The name "Bluetooth" is an Anglicized version of the Scandinavian Blåtand/Blåtann (Old Norse blátǫnn), the epithet of the tenth-century king Harald Bluetooth who united dissonant Danish tribes into a single kingdom and, according to legend, introduced Christianity as well. The idea of this name was proposed in 1997 by Jim Kardach of Intel who developed a system that would allow mobile phones to communicate with computers. At the time of this proposal he was reading Frans G. Bengtsson's historical novel *The Long Ships* about Vikings and King Harald Bluetooth. The implication is that Bluetooth does the same with communications protocols, uniting them into one universal standard.

Bluetooth is a wireless technology standard for exchanging data over short distances (using short-wavelength Ultra high frequency (UHF) radio waves in the industrial, scientific and medical (ISM) radio bands band from 2.4 to 2.485 GHz) from fixed and mobile devices, and building personal area networks (PANs). Invented by telecom vendor Ericsson in 1994, it was originally conceived as a wireless alternative to RS-232 data cables.

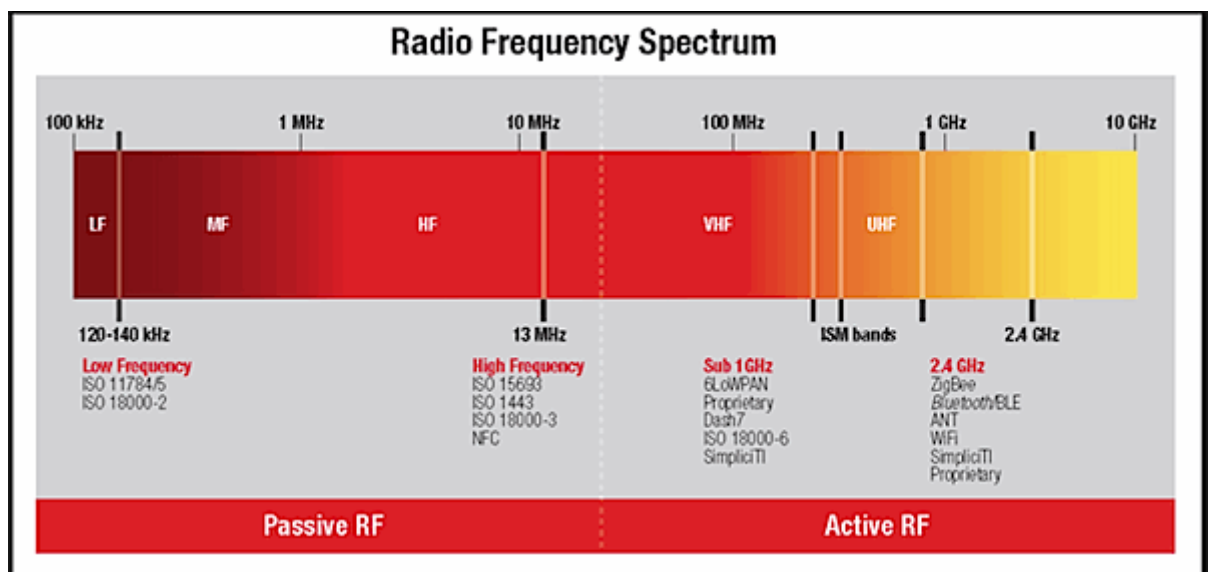


Figure 2.1 : Radio Frequency Spectrum (Wayne Staab 2013)

Bluetooth is managed by the Bluetooth Special Interest Group (SIG), which has more than 30,000 member companies in the areas of telecommunication, computing, networking, and consumer electronics. The IEEE standardized Bluetooth as IEEE 802.15.1, but no longer

maintains the standard. The Bluetooth SIG oversees development of the specification, manages the qualification program, and protects the trademarks. A manufacturer must meet Bluetooth SIG standards to market it as a Bluetooth device. A network of patents apply to the technology, which are licensed to individual qualifying devices.

Bluetooth operates at frequencies between 2402 and 2480 MHz, or 2400 and 2483.5 MHz including guard bands 2 MHz wide at the bottom end and 3.5 MHz wide at the top. This is in the globally unlicensed (but not unregulated) Industrial, Scientific and Medical (ISM) 2.4 GHz short-range radio frequency band. Bluetooth uses a radio technology called frequency-hopping spread spectrum. Bluetooth divides transmitted data into packets, and transmits each packet on one of 79 designated Bluetooth channels. Each channel has a bandwidth of 1 MHz. It usually performs 800 hops per second, with Adaptive Frequency-Hopping (AFH) enabled (Huang, Rudolph 2007).

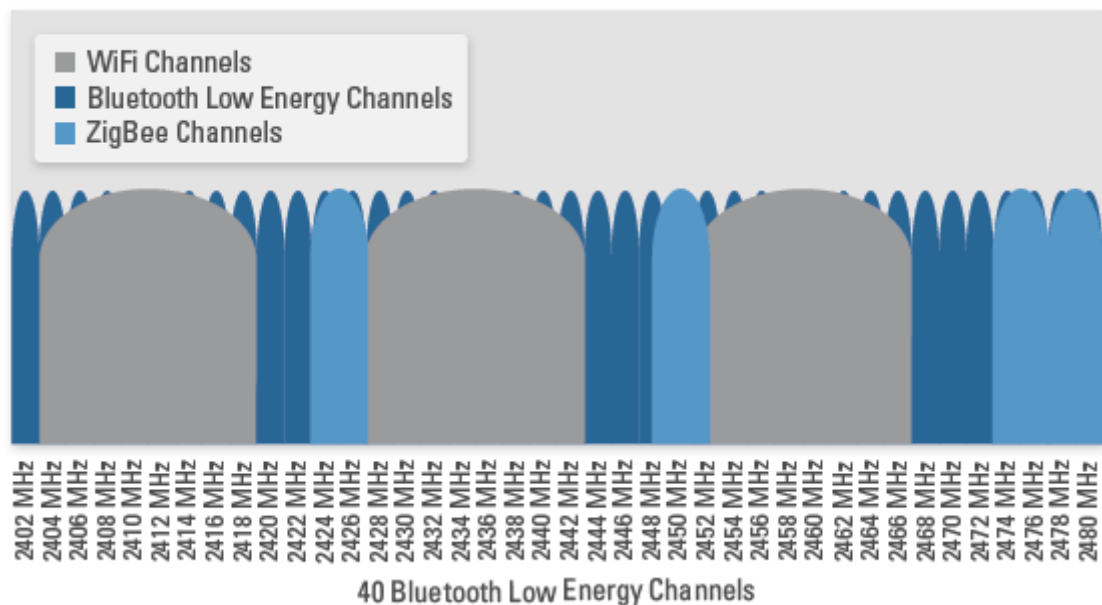


Figure 2.2: Bluetooth and WiFi Channels (Raajit Lall 2013)

Originally, Gaussian frequency-shift keying (GFSK) modulation was the only modulation scheme available. Since the introduction of Bluetooth 2.0+EDR, $\pi/4$ -DQPSK (Differential Quadrature Phase Shift Keying) and 8DPSK modulation may also be used between compatible devices. Devices functioning with GFSK are said to be operating in basic rate (BR) mode where an instantaneous data rate of 1 Mbit/s is possible. The term Enhanced Data Rate (EDR) is used to describe $\pi/4$ -DPSK and 8DPSK schemes, each giving 2 and 3 Mbit/s respectively. The combination of these (BR and EDR) modes in Bluetooth radio technology is classified as a "BR/EDR radio".

Bluetooth is a packet-based protocol with a master-slave structure. One master may communicate with up to seven slaves in a piconet. All devices share the master's clock. Packet exchange is based on the basic clock, defined by the master, which ticks at 312.5 μ s intervals. Two clock ticks make up a slot of 625 μ s, and two slots make up a slot pair of 1250 μ s. In the

simple case of single-slot packets the master transmits in even slots and receives in odd slots. The slave, conversely, receives in even slots and transmits in odd slots. Packets may be 1, 3 or 5 slots long, but in all cases the master's transmission begins in even slots and the slave's in odd slots.

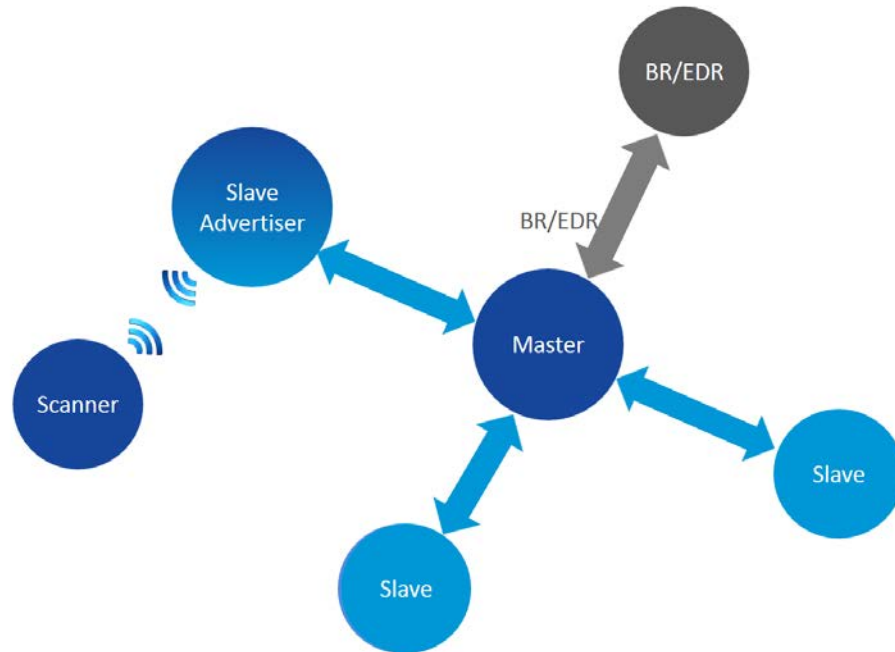


Figure 2.3: Master – Slave diagram (Hung Bui 2016)

2.1.2 Bluetooth class and version

2.1.2.1 Class definition

The first main characteristic that can classify bluetooth devices is called class, each device can run different class. For a start, I should make it clear that Bluetooth class 1, class 2 and class 3 all refer to Bluetooth power classes, not to be confused with Bluetooth device classes.

Table 2.1: Bluetooth classes (Bakker, McMichael Gilster 2002)

Class	Max. permitted power		Typ. range (m)
	(mW)	(dBm)	
1	100	20	100
2	2.5	4	10

3	1	0	1
4	0.5	-3	0.5

The effective range varies due to propagation conditions, material coverage, production sample variations, antenna configurations and battery conditions. Most Bluetooth applications are for indoor conditions, where attenuation of walls and signal fading due to signal reflections make the range far lower than specified line-of-sight ranges of the Bluetooth products. Most Bluetooth applications are battery powered Class 2 devices, with little difference in range whether the other end of the link is a Class 1 or Class 2 device as the lower powered device tends to set the range limit.

In some cases the effective range of the data link can be extended when a Class 2 device is connecting to a Class 1 transceiver with both higher sensitivity and transmission power than a typical Class 2 device. However, the Class 1 devices sensitivity is almost the same as Class 2 devices. Connecting two Class 1 devices with high sensitivity and high power can allow ranges far in excess of the typical 100m, depending on the requests required by the application. Some such devices allow open field ranges of up to 1 km and beyond between two similar devices without exceeding legal emission limits.

2.1.2.2 Bluetooth version

Bluetooth 1.0 and 1.1:

- Ratified as IEEE Standard 802.15.1–2002, many errors found in the v1.0B specifications were fixed.
- Added possibility of non-encrypted channels and received Signal Strength Indicator (RSSI).

Bluetooth 1.2:

- Adaptive frequency-hopping spread spectrum (AFH), which improves resistance to radio frequency interference by avoiding the use of crowded frequencies in the hopping sequence.
- Higher transmission speeds in practice than in v1.1, up to 721 kbit/s.
- Extended Synchronous Connections (eSCO), which improve voice quality of audio links by allowing retransmissions of corrupted packets, and may optionally increase audio latency to provide better concurrent data transfer.
- Host Controller Interface (HCI) operation with three-wire UART and Introduced Flow Control and Retransmission Modes for L2CAP.

Bluetooth 2.0 + EDR, 2.1 + EDR: The main difference is the introduction of an Enhanced Data Rate (EDR) for faster data transfer. The bit rate of EDR is 3 Mbit/s, although the maximum data transfer rate (allowing for inter-packet time and acknowledgements) is 2.1 Mbit/s.[45] EDR uses a combination of GFSK and Phase Shift Keying modulation (PSK) with two variants, $\pi/4$ -DQPSK and 8DPSK.[47] EDR can provide a lower power consumption through a reduced duty cycle.

The headline feature of v2.1 is secure simple pairing (SSP): this improves the pairing experience for Bluetooth devices, while increasing the use and strength of security.

Bluetooth 3.0 + HS provides theoretical data transfer speeds of up to 24 Mbit/s, though not over the Bluetooth link itself. Instead, the Bluetooth link is used for negotiation and establishment, and the high data rate traffic is carried over a collocated 802.11 link.

The main new feature is AMP (Alternative MAC/PHY), the addition of 802.11 as a high speed transport. The High-Speed part of the specification is not mandatory, and hence only devices that display the "+HS" logo actually support Bluetooth over 802.11 high-speed data transfer. Enables the use of alternative MAC and PHYs for transporting Bluetooth profile data. The Bluetooth radio is still used for device discovery, initial connection and profile configuration. However, when large quantities of data must be sent, the high speed alternative MAC PHY 802.11 (typically associated with Wi-Fi) transports the data. This means that Bluetooth uses proven low power connection models when the system is idle, and the faster radio when it must send large quantities of data.

Bluetooth 4.0 + LE, 4.1, 4.2: The Bluetooth SIG completed the Bluetooth Core Specification version 4.0 (called Bluetooth Smart) and has been adopted as of 30 June 2010. It includes Classic Bluetooth, Bluetooth high speed and Bluetooth low energy protocols. Bluetooth high speed is based on Wi-Fi, and Classic Bluetooth consists of legacy Bluetooth protocols. For further information of Low Energy Bluetooth, see chapter 2.2. Bluetooth 4.2, released on December 2, 2014; it Introduces features for the Internet of Things. The major areas of improvement are:

- Low Energy Secure Connection with Data Packet Length Extension.
- Link Layer Privacy with Extended Scanner Filter Policies.
- Internet Protocol Support Profile (IPSP) version 6 ready for Bluetooth Smart things to support connected home.

Bluetooth 5 has quadruple the range, double the speed, and provides an eight-fold increase in data broadcasting capacity of low energy Bluetooth transmissions compared to Bluetooth 4.x, which could be important for IoT applications where nodes are connected throughout a whole house. Bluetooth 5 supports transfers at 2 Mbit/s instead of the usual 1 Mbit/s. In addition it adds functionality for connectionless services like location-relevant information and navigation of low energy Bluetooth connections.

The major areas of improvement are:

- Slot Availability Mask (SAM).
- 2 Mbit/s PHY for LE
- LE Long Range
- High Duty Cycle Non-Connectable Advertising
- LE Advertising Extensions
- LE Channel Selection Algorithm #2

2.1.3 Protocol and connection

Bluetooth is defined as a layer protocol architecture consisting of core protocols, cable replacement protocols, telephony control protocols, and adopted protocols. Mandatory protocols for all Bluetooth stacks are: LMP, L2CAP and SDP. In addition, devices that communicate with Bluetooth almost universally can use these protocols: HCI and RFCOMM.

The heart of the Bluetooth specification is the Bluetooth protocol stack. By providing well-defined layers of functionality, the Bluetooth specification ensures interoperability of Bluetooth devices and encourages adoption of Bluetooth technology. As you can see in Figure 2.3, these layers range from the low-level radio link to the profiles.

At the base of the Bluetooth protocol stack is the radio layer. The radio module in a Bluetooth device is responsible for the modulation and demodulation of data into RF signals for transmission in the air. The radio layer describes the physical characteristics a Bluetooth device's receiver-transmitter component must have. These include modulation characteristics, radio frequency tolerance, and sensitivity level.

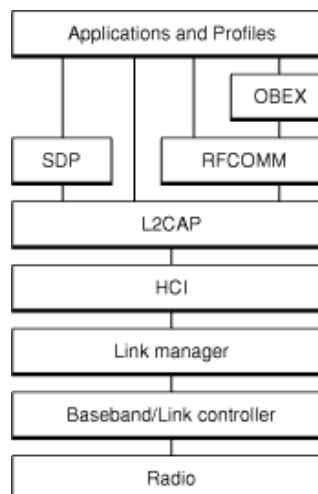


Figure 2.4: Bluetooth protocol stack

Above the radio layer is the baseband and link controller layer. The Bluetooth specification doesn't establish a clear distinction between the responsibilities of the baseband and those of the link controller. The best way to think about it is that the baseband portion of the layer is responsible for properly formatting data for transmission to and from the radio layer. In addition, it handles the synchronization of links. The link controller portion of this layer is responsible for carrying out the link manager's commands and establishing and maintaining the link stipulated by the link manager.

The link manager itself translates the host controller interface (HCI) commands it receives into baseband-level operations. It is responsible for establishing and configuring links and managing power-change requests, among other tasks.

Creating a Bluetooth connection between two devices is a multi-step process involving three progressive states:

- **Inquiry:** If two Bluetooth devices know absolutely nothing about each other, one must run an inquiry to try to discover the other. One device sends out the inquiry request, and any device listening for such a request will respond with its address, and possibly its name and other information.
- **Paging (Connecting):** Paging is the process of forming a connection between two Bluetooth devices. Before this connection can be initiated, each device needs to know the address of the other (found in the inquiry process).
- **Connection:** After a device has completed the paging process, it enters the connection state. While connected, a device can either be actively participating or it can be put into a low power sleep mode.
 - i. **Active Mode:** This is the regular connected mode, where the device is actively transmitting or receiving data.
 - i. **Sniff Mode:** This is a power-saving mode, where the device is less active. It'll sleep and only listen for transmissions at a set interval (e.g. every 100ms).
 - ii. **Hold Mode:** Hold mode is a temporary, power-saving mode where a device sleeps for a defined period and then returns back to active mode when that interval has passed. The master can command a slave device to hold.
 - iii. **Park Mode:** Park is the deepest of sleep modes. A master can command a slave to "park", and that slave will become inactive until the master tells it to wake back up.

When two Bluetooth devices share a special affinity for each other, they can be bonded together. Bonded devices automatically establish a connection whenever they're close enough. When I start up my car, for example, the phone in my pocket immediately connects to the car's Bluetooth system because they share a bond. No UI interactions are required.

Bonds are created through one-time a process called pairing. When devices pair up, they share their addresses, names, and profiles, and usually store them in memory. They also share a common secret key, which allows them to bond whenever they're together in the future.

Pairing usually requires an authentication process where a user must validate the connection between devices. The flow of the authentication process varies and usually depends on the interface capabilities of one device or the other. Sometimes pairing is a simple "Just Works" operation, where the click of a button is all it takes to pair (this is common for devices with no UI, like headsets). Other times pairing involves matching 6-digit numeric codes. Older, legacy (v2.0 and earlier), pairing processes involve the entering of a common PIN code on each

device. The PIN code can range in length and complexity from four numbers (e.g. "0000" or "1234") to a 16-character alphanumeric string (Gomez et al. 2012). For connection structure, see chapter 2.1.1.

2.2 Low Energy Bluetooth

2.2.1 Introduction to BLE

Bluetooth low energy (Bluetooth LE, BLE, marketed as Bluetooth Smart) is a wireless personal area network technology designed and marketed by the Bluetooth Special Interest Group aimed at novel applications in the healthcare, fitness, beacons, security, and home entertainment industries. Compared to Classic Bluetooth, Bluetooth Smart is intended to provide considerably reduced power consumption and cost while maintaining a similar communication range.

Bluetooth Smart was originally introduced under the name Wibree by Nokia in 2006. It was merged into the main Bluetooth standard in 2010 with the adoption of the Bluetooth Core Specification Version 4.0. Bluetooth Smart is not backward-compatible with the previous (often called "Classic") Bluetooth protocol. The Bluetooth 4.0 specification permits devices to implement either or both of the LE and Classic systems (Bluetooth developers 2016).

Bluetooth Smart uses the same 2.4 GHz radio frequencies as Classic Bluetooth, which allows dual-mode devices to share a single radio antenna. LE does, however, use a simpler modulation system.

The Bluetooth SIG identifies a number of markets for low energy technology, particularly in the smart home, health, sport and fitness sectors. Cited advantages include:

- Low power requirements, operating for "months or years" on a button cell.
- Small size and low cost.
- Compatibility with a large installed base of mobile phones, tablets and computers.

Borrowing from the original Bluetooth specification, the Bluetooth SIG defines several profiles — specifications for how a device works in a particular application — for low energy devices. Manufacturers are expected to implement the appropriate specifications for their device in order to ensure compatibility. A device may contain implementations of multiple profiles.

All current low energy application profiles are based on the generic attribute profile (GATT), a general specification for sending and receiving short pieces of data known as attributes over a low energy link. Bluetooth 4.0 provides low power consumption with higher bit rates.

In 2014, Cambridge Silicon Radio (now part of Qualcomm) launched CSR Mesh. CSR Mesh protocol uses Bluetooth Smart to communicate with other Bluetooth Smart devices in the network. Each device can pass the information forward to other Bluetooth Smart devices creating a “mesh” effect. For example, switching off an entire building of lights from a single smartphone.

2.2.2 Key Terms and Concepts

Summary of key BLE terms and concepts (Bluetooth developers 2016):

- **Generic Attribute Profile (GATT):** The GATT profile is a general specification for sending and receiving short pieces of data known as "attributes" over a BLE link. All current Low Energy application profiles are based on GATT.
- **The Bluetooth SIG defines many profiles for Low Energy devices.** A profile is a specification for how a device works in a particular application. Note that a device can implement more than one profile. For example, a device could contain a heart rate monitor and a battery level detector.
- **Attribute Protocol (ATT):** GATT is built on top of the Attribute Protocol (ATT). This is also referred to as GATT/ATT. ATT is optimized to run on BLE devices. To this end, it uses as few bytes as possible. Each attribute is uniquely identified by a Universally Unique Identifier (UUID), which is a standardized 128-bit format for a string ID used to uniquely identify information. The attributes transported by ATT are formatted as characteristics and services.
- **Characteristic:** A characteristic contains a single value and 0-n descriptors that describe the characteristic's value. A characteristic can be thought of as a type, analogous to a class.
- **Descriptor:** Descriptors are defined attributes that describe a characteristic value. For example, a descriptor might specify a human-readable description, an acceptable range for a characteristic's value, or a unit of measure that is specific to a characteristic's value.
- **Service:** A service is a collection of characteristics. For example, you could have a service called "Heart Rate Monitor" that includes characteristics such as "heart rate measurement." You can find a list of existing GATT-based profiles and services on bluetooth.org.

Here are the roles and responsibilities that apply when an Android device interacts with a BLE device:

- Central vs. peripheral: This applies to the BLE connection itself. The device in the central role scans, looking for advertisement, and the device in the peripheral role makes the advertisement.
- GATT server vs. GATT client: This determines how two devices talk to each other once they've established the connection.

To understand the distinction, imagine that you have an Android phone and an activity tracker that is a BLE device. The phone supports the central role; the activity tracker supports the peripheral role (to establish a BLE connection you need one of each—two things that only support peripheral couldn't talk to each other, nor could two things that only support central).

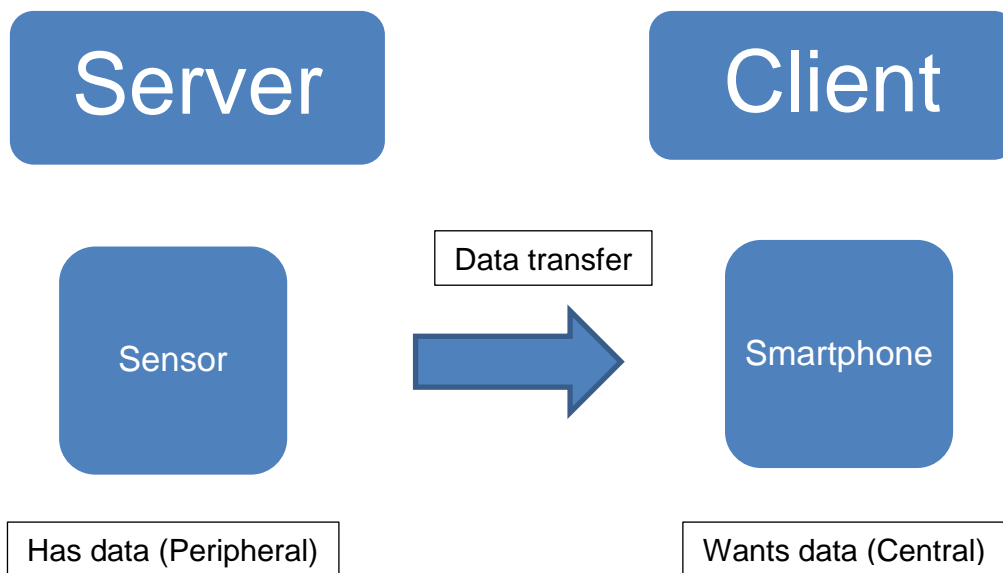


Figure 2.5: Server – Client Scheme

Once the phone and the activity tracker have established a connection, they start transferring GATT metadata to one another. Depending on the kind of data they transfer, one or the other might act as the server. For example, if the activity tracker wants to report sensor data to the phone, it might make sense for the activity tracker to act as the server. If the activity

tracker wants to receive updates from the phone, then it might make sense for the phone to act as the server.

2.2.3 GATT operations

The GATT protocol provides a number of commands for the client to discover information about the server. These include:

- Discover UUIDs for all primary services
- Find a service with a given UUID
- Find secondary services for a given primary service
- Discover all characteristics for a given service
- Find characteristics matching a given UUID
- Read all descriptors for a particular characteristic

Commands are also provided to read (data transfer from server to client) and write (from client to server) the values of characteristics:

- A value may be read either by specifying the characteristic's UUID, or by a handle value (which is returned by the information discovery commands above).
- Write operations always identify the characteristic by handle, but have a choice of whether or not a response from the server is required.
- 'Long read' and 'Long write' operations can be used when the length of the characteristic's data exceeds the MTU of the radio link.

Finally, GATT offers notifications and indications. The client may request a notification for a particular characteristic from the server. The server can then send the value to the client whenever it becomes available. For instance, a temperature sensor server may notify its client every time it takes a measurement. This avoids the need for the client to poll the server, which would require the server's radio circuitry to be constantly operational (Townsend 2015).

An indication is similar to a notification, except that it requires a response from the client, as confirmation that it has received the message. Now that we know a bit about BLE, we can see that there are two ways to get data from a device:

- Advertisement and Scan Responses: Advertisement include payload fields that can provide information, and the central device can request more information. There is no security inherent in this method because broadcasts are public, but multiple devices can receive the information at the same time.
- Connections: During a connection, two devices connect and exchange information, which allows much more information to be transmitted than using other methods.

2.3 Integrated Development Environment

2.3.1 Introduction to IDE

An integrated development environment (IDE) is a software application that provides comprehensive facilities to computer programmers for software development. An IDE normally consists of a source code editor, build automation tools and a debugger. Most modern IDEs have intelligent code completion. Some IDEs, such as Android Studio and Eclipse, contain a compiler, interpreter, or both; others, such as SharpDevelop and Lazarus, do not. The boundary between an integrated development environment and other parts of the broader software development environment is not well-defined. Sometimes a version control system, or various tools to simplify the construction of a Graphical User Interface (GUI), are integrated. Many modern IDEs also have a class browser, an object browser, and a class hierarchy diagram, for use in object-oriented software development.

Integrated development environments are designed to maximize programmer productivity by providing tight-knit components with similar user interfaces. IDEs present a single program in which all development is done. This program typically provides many features for authoring, modifying, compiling, deploying and debugging software.

One aim of the IDE is to reduce the configuration necessary to piece together multiple development utilities, instead providing the same set of capabilities as a cohesive unit. Reducing that setup time can increase developer productivity, in cases where learning to use the IDE is faster than manually integrating all of the individual tools. Tighter integration of all development tasks has the potential to improve overall productivity beyond just helping with setup tasks. For example, code can be continuously parsed while it is being edited, providing instant feedback when syntax errors are introduced. That can speed learning a new programming language and its associated libraries.

Some IDEs are dedicated to a specific programming language, allowing a feature set that most closely matches the programming paradigms of the language. However, there are many multiple-language IDEs.

While most modern IDEs are graphical, text-based IDEs such as Turbo Pascal were in popular use before the widespread availability of windowing systems like Microsoft Windows and the X Window System (X11). They commonly use function keys or hotkeys to execute frequently used commands or macros (Jesse Russell 2012).

2.3.2 Android Studio

Android Studio is the official integrated development environment (IDE) for the Android platform. It was announced on May 16, 2013 at the Google I/O conference.

Android Studio was in early access preview stage starting from version 0.1 in May 2013, then entered beta stage starting from version 0.8 which was released in June 2014. The first stable build was released in December 2014, starting from version 1.0.

Based on JetBrains' IntelliJ IDEA software, Android Studio is designed specifically for Android development. It is available for download on Windows, macOS and Linux, and replaced

Eclipse Android Development Tools (ADT) as Google's primary IDE for native Android application development.

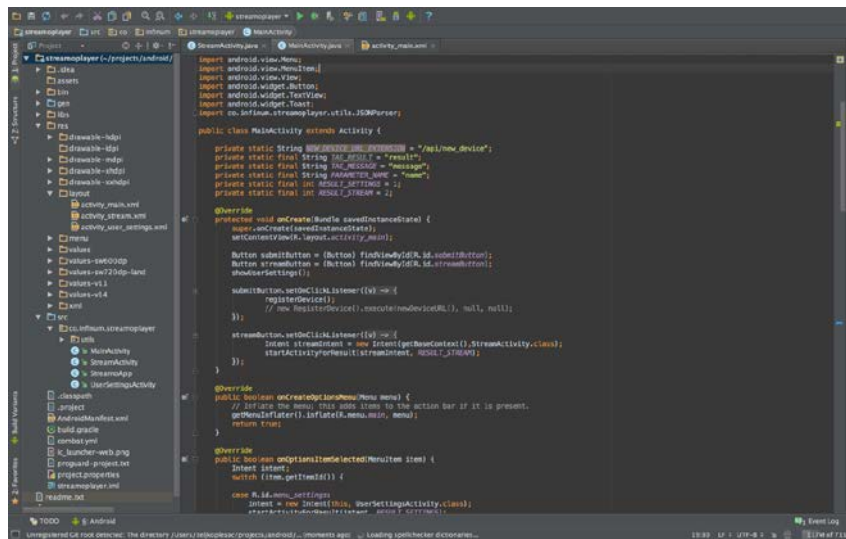


Figure 2.6: Android Studio GUI (Željko Plesac 2013)

Android Studio offers even more features that enhance productivity when building Android apps, such as:

- A flexible Gradle-based build system
- A fast and feature-rich emulator
- A unified environment where you can develop for all Android devices
- Instant Run to push changes to your running app without building a new APK
- Code templates and GitHub integration to help you build common app features and import sample code
- Extensive testing tools and frameworks
- Lint tools to catch performance, usability, version compatibility, and other problems
- C++ and NDK support
- Built-in support for Google Cloud Platform, making it easy to integrate Google Cloud Messaging and App Engine

Android Studio is certainly a step ahead of Eclipse, which lost its position in less than a year as the main IDE for android application development and became died out. Also, as

Android Studio has been developed by android programmers, has much more compatibility with OS and specific tools (Yener, Dundar 2016). Further comparison shown in chapter 4.2.

2.3.3 Java Language

Java is a general-purpose computer programming language that is concurrent, class-based, object-oriented, and specifically designed to have as few implementation dependencies as possible. It is intended to let application developers "write once, run anywhere" (WORA), meaning that compiled Java code can run on all platforms that support Java without the need for recompilation. Java applications are typically compiled to bytecode that can run on any Java virtual machine (JVM) regardless of computer architecture. As of 2016, Java is one of the most popular programming languages in use, particularly for client-server web applications, with a reported 9 million developers. Java was originally developed by James Gosling at Sun Microsystems (which has since been acquired by Oracle Corporation) and released in 1995 as a core component of Sun Microsystems' Java platform. The language derives much of its syntax from C and C++, but it has fewer low-level facilities than either of them.

The original and reference implementation Java compilers, virtual machines, and class libraries were originally released by Sun under proprietary licenses. As of May 2007, in compliance with the specifications of the Java Community Process, Sun relicensed most of its Java technologies under the GNU General Public License. Others have also developed alternative implementations of these Sun technologies, such as the GNU Compiler for Java (bytecode compiler), GNU Classpath (standard libraries), and IcedTea-Web (browser plugin for applets) (Hoisington 2015).

When we consider a Java program, it can be defined as a collection of objects that communicate via invoking each other's methods. Let us now briefly look into what do class, object, methods, and instance variables mean.

- **Object:** Objects have states and behaviors. Example: A dog has states - color, name, breed as well as behavior such as wagging their tail, barking, eating. An object is an instance of a class.
- **Class:** A class can be defined as a template/blueprint that describes the behavior/state that the object of its type supports.
- **Methods:** A method is basically a behavior. A class can contain many methods. It is in methods where the logics are written, data is manipulated and all the actions are executed.
- **Instance Variables:** Each object has its unique set of instance variables. An object's state is created by the values assigned to these instance variables.

Let us look at a simple code that will print the words Hello World:

```
public class MyFirstJavaProgram {  
  
    /* This is my first java program.  
     * This will print 'Hello World' as the output  
     */  
  
    public static void main(String []args) {  
        System.out.println("Hello World"); // prints Hello World  
    }  
}
```

This simple code make this output in the system:

```
C:\> javac MyFirstJavaProgram.java  
C:\> java MyFirstJavaProgram  
Hello World
```

When we start to develop software based on java, we must keep in mind this important points about java logic:

- **Case Sensitivity:** Java is case sensitive, which means identifier Hello and hello would have different meaning in Java.
- **Class Names:** For all class names the first letter should be in Upper Case. If several words are used to form a name of the class, each inner word's first letter should be in Upper Case. Example: class MyFirstJavaClass.
- **Method Names:** All method names should start with a Lower Case letter. If several words are used to form the name of the method, then each inner word's first letter should be in Upper Case. Example: public void myMethodName().
- **Program File Name:** Name of the program file should exactly match the class name.
- **When saving the file,** you should save it using the class name (Remember Java is case sensitive) and append '.java' to the end of the name (if the file name and the class name do not match, your program will not compile). Example: Assume 'MyFirstJavaProgram' is the class name. Then the file should be saved as 'MyFirstJavaProgram.java'
- **public static void main(String args[]):** Java program processing starts from the main() method which is a mandatory part of every Java program.

An Android library is structurally the same as an Android app module. It can include everything needed to build an app, including source code, resource files, and an Android manifest. However, instead of compiling into an APK that runs on a device, an Android library compiles into an Android Archive (AAR) file that you can use as a dependency for an Android app module. Unlike JAR files, AAR files can contain Android resources and a manifest file, which allows you to bundle in shared resources like layouts and drawables in addition to Java classes and methods (Liu 2013).

A library module is useful in the following situations:

- When you're building multiple apps that use some of the same components, such as activities, services, or UI layouts.
- When you're building an app that exists in multiple APK variations, such as a free and paid version and you need the same core components in both.
- In either case, simply move the files you want to reuse into a library module then add the library as a dependency for each app module. This page teaches you how to do both.

2.3.4 XML Language in Android

A layout defines the visual structure for a user interface, such as the UI for an activity or app widget. You can declare a layout in two ways:

- Declare UI elements in XML. Android provides a straightforward XML vocabulary that corresponds to the View classes and subclasses, such as those for widgets and layouts.
- Instantiate layout elements at runtime. Your application can create View and ViewGroup objects (and manipulate their properties) programmatically.

The Android framework gives you the flexibility to use either or both of these methods for declaring and managing your application's UI. For example, you could declare your application's default layouts in XML, including the screen elements that will appear in them and their properties. You could then add code in your application that would modify the state of the screen objects, including those declared in XML, at run time.

You should also try the Hierarchy Viewer tool, for debugging layouts — it reveals layout property values, draws wireframes with padding/margin indicators, and full rendered views while you debug on the emulator or device.

The layoutopt tool lets you quickly analyze your layouts and hierarchies for inefficiencies or other problems.

The advantage to declaring your UI in XML is that it enables you to better separate the presentation of your application from the code that controls its behavior. Your UI descriptions are external to your application code, which means that you can modify or adapt it without having to modify your source code and recompile. For example, you can create XML layouts for different screen orientations, different device screen sizes, and different languages. Additionally, declaring the layout in XML makes it easier to visualize the structure of your UI, so it's easier to debug problems. As such, this document focuses on teaching you how to declare your layout in XML. If you're interested in instantiating View objects at runtime, refer to the ViewGroup and View class references.

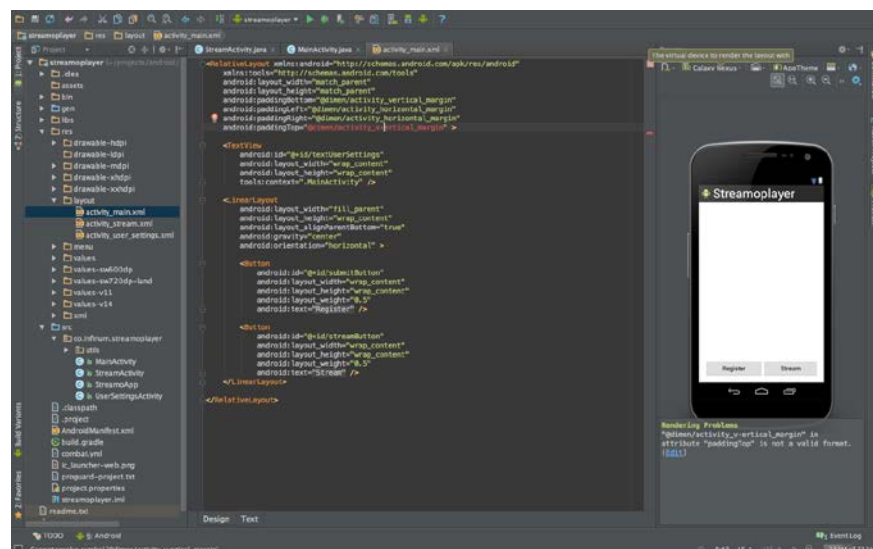


Figure 2.7: XML design in Android Studio (Željko Plesac 2013)

In general, the XML vocabulary for declaring UI elements closely follows the structure and naming of the classes and methods, where element names correspond to class names and attribute names correspond to methods. In fact, the correspondence is often so direct that you

can guess what XML attribute corresponds to a class method, or guess what class corresponds to a given XML element. However, note that not all vocabulary is identical. In some cases, there are slight naming differences. For example, the `EditText` element has a `text` attribute that corresponds to `EditText.setText()`.

Using Android's XML vocabulary, you can quickly design UI layouts and the screen elements they contain, in the same way you create web pages in HTML — with a series of nested elements.

Each layout file must contain exactly one root element, which must be a `View` or `ViewGroup` object. Once you've defined the root element, you can add additional layout objects or widgets as child elements to gradually build a `View` hierarchy that defines your layout. For example, here's an XML layout that uses a vertical `LinearLayout` to hold a `TextView` and a `Button`:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical" >
    <TextView android:id="@+id/text"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello, I am a TextView" />
    <Button android:id="@+id/button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello, I am a Button" />
</LinearLayout>
```

3 State of the art

3.1 Bluetooth solution

Nowadays, industry 4.0 is being developed as it is a phenomena born few years ago. Bluetooth solutions are still being developed by cutting edge enterprises. One state of the art which is bluetooth based is the following, but is thought to communicate between a bluetooth smart tool machine and a computer, Android application is not supported. The information flow is the following:

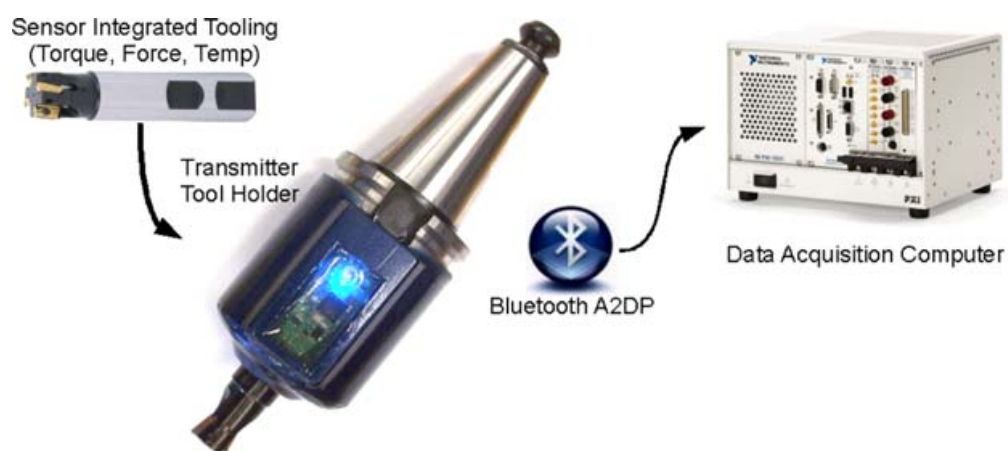


Figure 3.1 Block diagram of wireless sensor system (Suprock, Nichols 2009)

The amplifier and transmitter circuits are built into an ABS plastic shroud mounted on the exterior of a C40 set screw type tool holder. The tool holder was modified to house the female mini DIN connector and route signal cables to the amplifier circuit. No modifications were made to alter the compliance of the tool holder body. Figure 1 shows the tool holder with the amplifier circuit in the ABS plastic shroud and a block diagram of the data route.

In an end milling system, significant challenges to wireless bandwidth may exist from motor noise. During the machining process, spindle and bed motors generate a wide and continually changing spectrum of interference. This may increase the difficulty of wireless transmission with fixed frequency or amplitude methods. The problem can be approached in two ways: Characterizing the spectrum of motor noise and designing a transmitter to avoid it for a particular milling process, or, employing an active noise avoidance scheme such as Frequency Hopping Spread Spectrum (FHSS) as shown in chapter 2.1.

High quality audio transmission has been at the forefront of this technology with interoperability between all Bluetooth audio devices being based on the Audio Distribution Model (Bluetooth SIG, 2007). Transmission bandwidth requirements conform to mandatory sampling

requirements of 44.1 and 48.0 kHz. These rates are enforced for the benefit of the transmitter by the receiver device (Bluetooth SIG, 2007).

For this work, a commercially available 16 bit audio transmitter is used. The transmitter circuit is modified from a Com One A2DP stereo audio transmitter (retail cost approximately \$40). The bridge circuit is located on the outer radius of the end mill cutting tool. An indexable insert cutter, ISCAR HELI2000, was chosen for this work. This cutter represents a 'worst case' scenario for the measurement of strain. It is a short overhang tool with a 19.05 mm shank. However, it was estimated that wire gages would provide sufficient sensitivity for a torsion bridge.

The relationship for sensitivity can be given as:

$$\frac{V_o}{V} = \frac{FTD_o}{2IoE} \quad \text{Equation 3.1}$$

where V_o is the bridge output voltage, V is the excitation voltage, F is the gage factor, E is the material modulus, D_o is the tool diameter and I_o is the area moment of inertia. At a 3 volt excitation voltage, the sensitivity is expected to be approximately $4.3 \times 10^{-5} \text{ V/ (N*m)}$. This is acceptable, considering the amplifier and software gains. With a 100x amplifier gain and a 10x software gain, the recorded voltage will reach 1 volt when 23.2 N*m is applied to the tool.

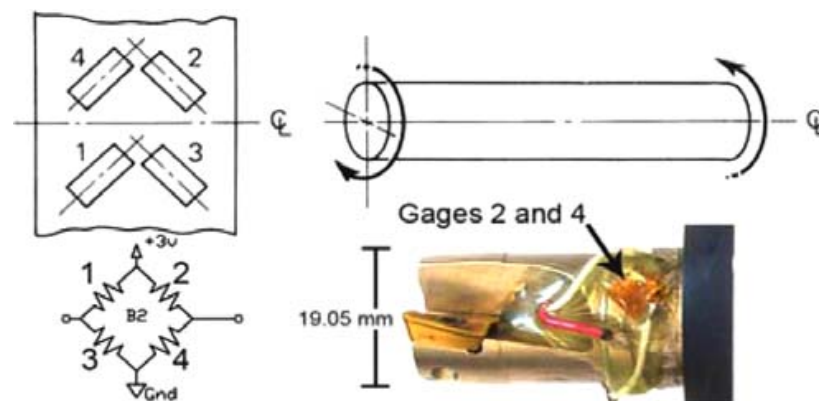


Figure 3.2: Torque bridge on cutting tool with electrical diagram of gage bridge and schema (Suprock, Nichols 2009)

The tool used is a through-coolant design. The coolant channel provided a convenient route for the sensor signal wires without interrupting the cutting process or altering the physical geometry of the tool holder. Figure 3.2 details the position of the mini DIN connector and strain gage rosettes. Because of the prototype nature of this tool, the gages and signal wires were protected by epoxy and not fully encased by a protective cover.

Following a similar design scheme to the vibration sensor-integrated tooling systems described in Suprock et al. (2008a), this high bandwidth stereo transmitter includes a printed circuit amplifier board, symmetric geometry for low eccentricity and reduced wire routing for an improved signal to noise ratio. This prototype is also waterproof for experimental testing with cutting fluid. Figure 6 shows a profile of the transmitter tool holder design featuring the amplifier circuit facet:

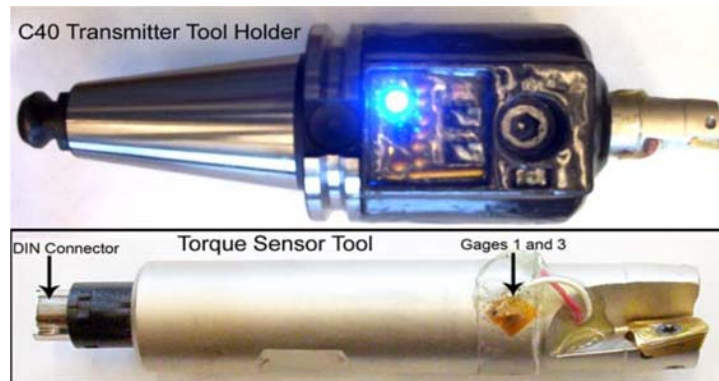


Figure 3.3: : Torque sensor integrated tool and transmitter (Suprock, Nichols 2009)

As there is no a formal solution about this topic with bluetooth (are still being developed) we pass to the next technology.

3.2 Wi-Fi & ZigBee solutions

Wi-Fi solutions in this field are not as useful as bluetooth because the energy that it needs to run is higher than Bluetooth and much more compared with BLE as shown in Figure 3.4:

ZigBee, Bluetooth, NFC, vs., WiFi

	Low Energy Bluetooth	ZigBee	NFC	Low Power WiFi
Frequency (MHz)	2402 – 2482	868 - 868.8, 902 - 928, 2402 – 2482	13.56	2400 - 2500
Channels	3	16	1	3
Modulation	GFSK	BPSK & QPSK	ASK	64QAM
Max potential data rate	1 Mbps	250 Kbps	424 Kbps	54 Mbps
Range	10m	100+m	10cm	30m
Power Profile	Days	Months/Years	Months/Years	Hours
Complexity	Complex	Simple	Simple	Complex
Nodes/Master	7	65,000	1+1	
Extendibility	No	Yes	No	Yes ³⁷

Figure 3.4: : Comparison between wireless communication tech. (Johen Schiler 2015)

Our case of study, in the end, is just the communication between a sensor and a smartphone. This is widely studied with Wi-Fi technology in smart home management. Home automation or smart home (also known as domotics) is building automation for the home. It involves the control and automation of lighting, heating (such as smart thermostats), ventilation, air conditioning (HVAC), and security, as well as home appliances such as washer/dryers, ovens or refrigerators/freezers. Wi-Fi is often used for remote monitoring and control. Home devices, when remotely monitored and controlled via the Internet, are an important constituent of the Internet of Things. Modern systems generally consist of switches and sensors connected to a central hub sometimes called a "gateway" from which the system is controlled with a user interface that is interacted either with a wall-mounted terminal, mobile phone software, tablet computer or a web interface, often but not always via Internet cloud services.

According to Li et al. (2016) there are three generations of home automation:

- First generation: wireless technology with proxy server, e.g. Wi-Fi automation;
- Second generation: artificial intelligence controls electrical devices, e.g. amazon echo;
- Third generation: robot buddy "who" interacts with humans, e.g. Robot Rovio, Roomba.



Figure 3.5: Smart house example interface

The word "domotics" (and "domotica" when used as a verb) is a contraction of the Latin word for a home (domus) and the word robotics. Applications and technologies used:

- Heating, ventilation and air conditioning (HVAC): it is possible to have remote control of all home energy monitors over the internet incorporating a simple and friendly user interface.
- Lighting control system
- Occupancy-aware control system: it is possible to sense the occupancy of the home using smart meters and environmental sensors like CO₂ sensors, which can be integrated into the building automation system to trigger automatic responses for energy efficiency and building comfort applications.
- Appliance control and integration with the smart grid and a smart meter, taking advantage, for instance, of high solar panel output in the middle of the day to run washing machines.
- Security: a household security system integrated with a home automation system can provide additional services such as remote surveillance of security cameras over the Internet, or central locking of all perimeter doors and windows.
- Leak detection, smoke and CO detectors.
- Indoor positioning systems.
- Home automation for the elderly and disabled.

Despite this seems far from our thesis, it faces the same objective as we want to develop an application for showing the received data from the sensor through the bluetooth module.

On the other hand, we have ZigBee technology. ZigBee is an IEEE 802.15.4-based specification for a suite of high-level communication protocols used to create personal area networks with small, low-power digital radios, such as for home automation, medical device data

collection, and other low-power low-bandwidth needs, designed for small scale projects which need wireless connection.



Figure 3.6: ZigBee module

Its low power consumption limits transmission distances to 10–100 meters line-of-sight, depending on power output and environmental characteristics. ZigBee devices can transmit data over long distances by passing data through a mesh network of intermediate devices to reach more distant ones. ZigBee is typically used in low data rate applications that require long battery life and secure networking (ZigBee networks are secured by 128 bit symmetric encryption keys.) ZigBee has a defined rate of 250 kbit/s, best suited for intermittent data transmissions from a sensor or input device.

A state from the art is implementation of can and ZigBee networks based Industrial monitoring and control applications (K.Pavani 2012). Connection of the hardware shown below in Figure 3.7.

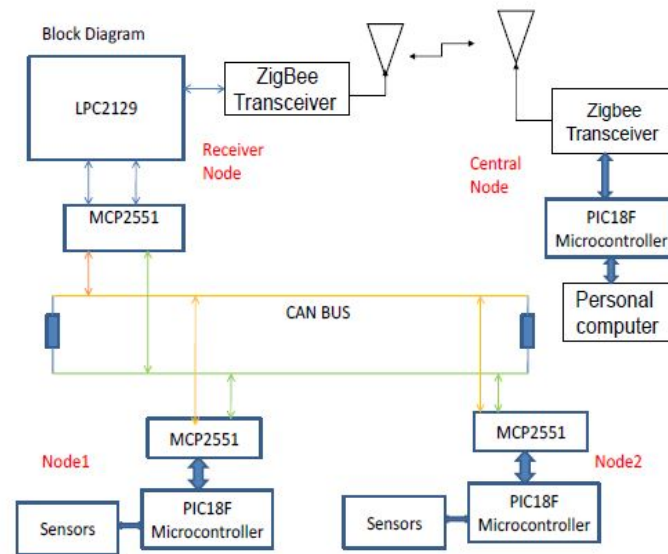


Figure 3.7: Block diagram for monitoring and controlling

Sensors used in that research: we use the various sensors usually designed by electronics components for sensing the industrial parameters like temperature, pressure, fire or smoke, intensity levels, humidity, etc. Temperature sensor as shown in figure 3.8 and gas sensor in figure 3.9.

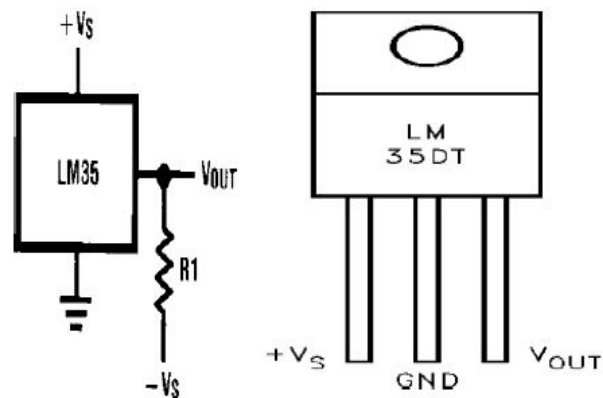


Figure 3.8 : Temperature sensor

Calibrated directly in ° Celsius (Centigrade) Linear variation of +10.0 mV/°C This 0.5°C accuracy guaranteed (at +25°C) Rated for full -55° to +150°C range Low cost Operates from 4 to 30 volts

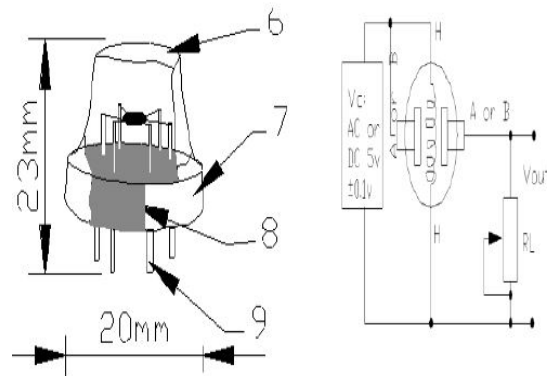


Figure 3.9: Gas sensor

Gas sensors are High sensitivity to LPG, iso-butane, propane and less sensitivity to alcohol, smoke. They are used in gas leakage detecting equipment in family and industry, are suitable for detecting of LPG, iso-butane. The conclusion of this studio was successful achievement of the objectives and also, in order to make those products compatible with equipment from other sources, we plan to migrate from the present CANbus proprietary protocol to the CANopen standard. In this new framework, Zigbee Application Layer features are to be used, therefore providing full compatibility in international markets.

3.3 Summary

Having an overall view of industrial solutions that is already in the industrial world, we can say that bluetooth or ZigBee development focused on machining must be developed. ZigBee installations are not as friendly as Bluetooth, as Bluetooth can be implemented in wearable devices as Smartphones, Tablets and even watches.



Figure 3.10: Smartwatch GPS sensor

In the future, an industrial worker could control his machining machine with his watch, receiving the data from the sensors that are implemented in the machine through a bluetooth module and a motherboard that process the information, this is beginning to be developed in smart houses as seen with Wi-Fi but, as homes are thought to be static and non-power dependent as could happen in a smart tool were electronical circuits feeding and durability is a hard challenge, Wi-Fi is used.

4 Own approach

4.1 Bluetooth module selection

To achieve our objectives, we have to select a bluetooth module which is going to be used to send the data from the Arduino which is controlling the smart tool. First of all, we have to discuss what technology (bluetooth classic or bluetooth low energy), bluetooth class and bluetooth version.

In our case, as the whole sensor device will be placed in the jaw without any feeding link, battery consumption is the main parameter we have to focus on. Furthermore, we have to take in care the over the air data rate (OADR) as this is also an important parameter. In table 4.1, we see a comparison of both technologies:

Table 4.1: Bluetooth classic vs BLE

Technical Specification	Classic bluetooth	Bluetooth low energy
Range (Class dependent, av.)	100 m (330 ft)	50 m (160 ft)
Over the Air Data Rate	1 – 3 Mbit/s	1 Mbit/s
Application Throughput	0.7 – 2.1 Mbit/s	1 Mbit/s
Active Slaves	7	Flexible
	56 / 128-bit and	128-bit AES with Counter

Security	application layer user defined	Mode CBC-MAC and application layer user defined
Robustness	Adaptive fast frequency hopping, FEC, fast ACK	Adaptive frequency hopping, Lazy Acknowledgement, 24-bit CRC, 32-bit Message Integrity Check
Latency (from a non- connected state)	Typically 100 ms	6 ms
Total time to send data (depending battery life)	100 ms	3 ms, less than 3 ms
Network topology	Scatternet	Star-bus
Power consumption	1 as the reference	0.01 to 0.5 (depending on the use case)
Peak current consumption	Less than 30 mA	Less than 20 mA

As seen, our main important parameters depend more on the chip than in the technology, so we compare to different commercial chips that are very common in electronic projects : HC-05 and HM-10, this last one is the Low Energy module.

4.1.1 HC-05

HC-05 module is an easy to use Bluetooth SPP (Serial Port Protocol) module, designed for transparent wireless serial connection setup by Linvor's (Formerly known as Guangzhou HC Information Technology Co., Ltd; Chinese enterprise focused on wireless communication hardware).

Serial port Bluetooth module is fully qualified Bluetooth V2.0+EDR (Enhanced Data Rate) 3Mbps Modulation with complete 2.4GHz radio transceiver and baseband. It uses CSR Bluecore 04-External single chip Bluetooth system with CMOS technology and with AFH (Adaptive Frequency Hopping Feature). It has the footprint as small as 12.7mmx27mm, this will simplify your overall design/development cycle. For further information about Bluetooth specifications, see chapter 2.1. In the following figure, we see the module:

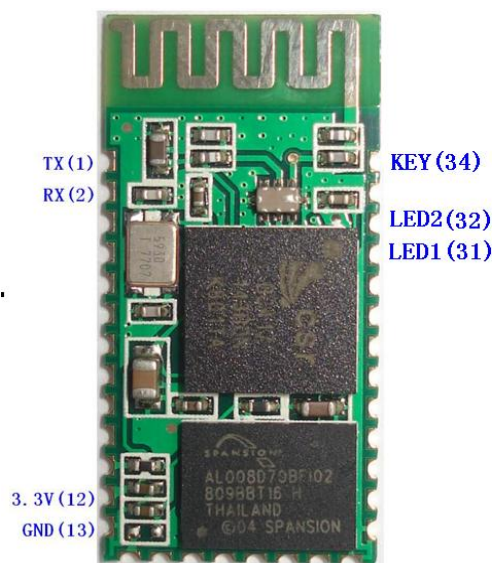


Figure 4.1: HC-05 Module

This module has the following hardware main features, for further information, check Annex A (HC-05 official datasheet provided by manufacture enterprise):

- Typical -80dBm sensitivity.
- Class 2: Up to +4dBm RF transmit power.
- Low Power 1.8V Operation, 1.8 to 3.6V I/O.
- PIO control.
- ART interface with programmable baud rate.
- With integrated antenna.
- With edge connector

Now we have to check the consumption from the batteries. For checking that, we have to check it in the HC-05 User Instructional Manual (Annex B) in which consumption feature is shown as: During the pairing, the current is fluctuant in the range of 30-40mA. The mean current is about 25mA. There is no sleep mode. This parameter is same for all the Bluetooth modules (HC device family).

For further information about the electronic diagram, PIN information, component size, go to Annex A & B.

4.1.2 HM-10

HM-10, developed by JNHuaMao Technology Company and based on the Texas Instruments' CC2541 system on chip. HM-10 encapsulates a serial connection that works over two of its pins, so it can be connected to the microcontroller's UART serial port without any additional configurations. In the HM-10 module datasheet (Annex C) we can find the following characteristics:

- Bluetooth Specification V4.0 BLE
- Send and receive no bytes limit.
- Working frequency: 2.4GHz ISM band
- Modulation method: GFSK(Gaussian Frequency Shift Keying)
- RF Power: -23dbm, -6dbm, 0dbm, 6dbm, can modify through AT
- Speed: Asynchronous: 6K Bytes
- Synchronous: 6K Bytes
- Security: Authentication and encryption
- Service: Central & Peripheral UUID FFE0,FFE1
- Power: +3.3VDC 50mA
- Long range: Open space have 100 Meters with iphone4s
- Power: In sleep mode 400uA~1.5mA, Active mode 8.5mA.
- Working temperature: -5 ~ +65 Centigrade
- Size: HM- 10 26.9mm x 13mm x 2.2 mm; HM-11 18*13.5*2.2mm

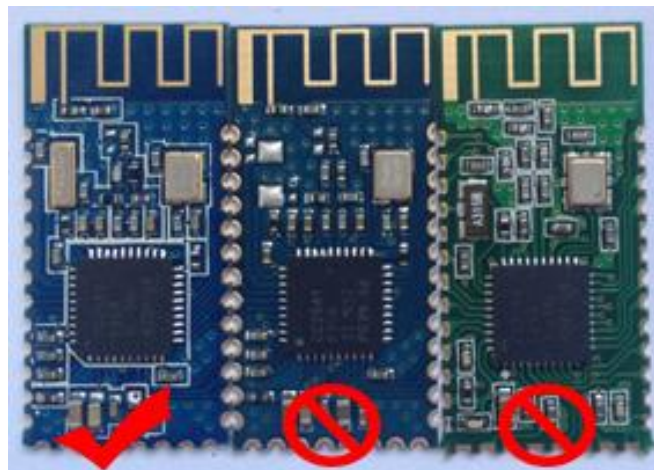


Figure 4.2 HM-10 Official vs fake

According with what we are looking for, we see that it uses Bluetooth Specification V4.0 BLE and a consumption in active mode of 8.5mA; the other main specification, bluetooth class, is defined in the CC2541 datasheet by TI (Annex D), where we find that it is Class 2. For further specifications about bluetooth and electronic diagrams about HM-10, see Annex C & D.

The following information is very important to understand how the application connects and communicate with the device (in our case, Arduino controller). The HM-10 abstracts and packs a Bluetooth Low Energy connection in a serial connection. The original out-of-the-box firmware of the module exposes a BLE peripheral with a proprietary connectivity service (Service UUID: 0000ffe0-0000-1000-8000-00805f9b34fb) that enables bidirectional communication between the module and any other central device that connects to it. The service defines a single characteristic (Characteristic UUID: 0000ffe1-0000-1000-8000-00805f9b34fb) that stores 20 bytes of unformatted data:

- When the central device wants to send data to the module, it WRITES the characteristic with the desired content
- When the module wants to send data, it sends a NOTIFICATION to the central device

The HM-10 module implements a serial connection in pin 1 (TXD in breakout boards) and pin 2 (RXD) that is linked logically to the BLE service and connection. Any data that is received through the RXD pin is sent through notifications to the central device. Any data written by the central device is output through the TXD pin. This mechanism wraps the BLE connection as a standard serial connection for the connected microcontroller (Arduino, Raspberry Pi).

Despite ISM is shared with ZigBee and Wi-Fi, interferences cannot take part because of its modulation: GFSK (Gaussian Frequency Shift Keying).

4.1.3 Conclusions

HM-10 is two steps ahead HC-05 as it uses more modern technology (v4.0 vs 2.0 + EDR) with best features as shown in chapter 2.1., lower consumption in active mode and HM-10 has

sleep mode to achieve even lower average consumption but both modules have same range (Class 2).

Table 4.2 Specifications Summary

Specification	HM-10	HC-05
Bluetooth version	Bluetooth LE 4.0	Bluetooth 2.0 + EDR
Active consumption (mA)	8.5 mA	25mA
Bluetooth class	2	2
Sleep Mode	Yes	No

For this project, we finally decide to use HM-10 because of consumption with the same range of action as is the main design parameter of the sensor device located in the jaws and newer bluetooth version as shown.

4.2 Operating system & IDE selection

4.2.1 Operating system selection

Decide what operating system we are going to focus in is a very important issue as it changes the IDE we use, programming language, target groups, etc.

The Android operative system was chosen due to a few reasons. The most important one is the fact that, as of the first quarter of 2016, Android holds 84.1% market share, with an astonishing number of over 293 million sales in that period.

Table 4.3: Worldwide Smartphone Sales to End Users by Operating System in 4Q16, Thousands of Units (Gartner 2017)

Operating System	1Q16 Units	1Q16 Market Share (%)	1Q15 Units	1Q15 Market Share (%)
------------------	---------------	--------------------------	---------------	--------------------------

Android	293,771.2	84.1	264,941.9	78.8
iOS	51,629.5	14.8	60,177.2	17.9
Windows	2,399.7	0.7	8,270.8	2.5
Blackberry	659.9	0.2	1,325.4	0.4
Others	791.1	0.2	1,582.5	0.5
Total	349,251.4	100.0	336,297.8	100.0

The use of the Java programming language also made the choice of Android easier as its Android's native language, giving its features when it comes to high-level programming. When low-level operations are needed, a library that implement them is always found, and the graphic management of the app is also easily implemented with XML language.

Although Bluetooth Low Energy is supported since API (Application programming interface) level 18, minimum target API level 21 was chosen due to programming reasons. API level 21, or Android 5.0 – 5.0.2 Lollipop and further versions hold 45.5% penetration on Android devices as of June 2016, so a big amount of these devices would be able to install the app. This number is growing so fast that it already may be bigger than 50%.

Table 4.4 Android version market share 2016 (Android developers 2017)

Version	Codename	API	Distribution
2.2	Froyo	8	0.1%
2.3.3 - 2.3.7	Gingerbread	10	2.0%
4.0.3 - 4.0.4	Ice-cream Sandwich	15	1.9%
4.1.x	Jelly Bean	16	6.8%
4.2.x		17	9.4%
4.3		18	2.7%
4.4	KitKat	19	31.6%
5.0	Lollipop	21	15.4%
5.1		22	20.0%
6.0	Marshmallow	23	10.1%

Furthermore, having an industrial economic view, it is much cheaper Android than iOS and there are lot of different products to work with and so you choose the device that fits with your solution meanwhile if you decide to work with iOS, you must work with iPhone or iPad.

4.2.2 IDE selection

As we have to choose an environment to develop our app, we have two main options: Android Studio vs Eclipse. Both are Java based and have huge community behind. We compare the two IDEs in 6 distinct areas to reveal why Android Studio is a step ahead of Eclipse (Grant, Haseman 2014):

- **Gradle Integration:** Android Studio uses the quick growing Gradle build system that is so integrated, and Gradle is really a great tool. If you have decided to go with Eclipse then yet say to look at Gradle's features and try it out and see if it fits with your project. In case you want to go with Android Studio, no need to worry about being stuck with Gradle system because it is really good. Eclipse uses Apache Ant as its prime build system that is an extremely robust XML based build system and lots of Java developers have been already familiar with it.
- **Advanced Code Completion:** Both Android Studio and Eclipse feature the typical Java code auto completion. But, we usually found that the code completion is really better on AS compare to Eclipse which looks to get a bit perplexed at times and doesn't provide precise results most of the time. Keep in mind, the more time you will spend as a programmer grinding out code, the more you value code completion.
- **User Interface (UI):** We know Eclipse interface and quirks very well, it is big and somewhat overwhelmed, but we have to face it because most IDEs are overwhelming when you use them first time. So, keeping this in mind and found that the tools and menu items in Android Studio tend to get me where we want to be a little more promptly and effortlessly than their counterparts in Eclipse. In addition, AS was built purposely for Android, while Eclipse was built to all-purpose IDE that can be used with any language and platform.
- **Organization of Project:** Although, both IDEs work in a different way to help you manage and organize your projects, but when you want to work on many projects in Eclipse you need to merge them into a workspace. In an attempt to switch to a different workspace, you have to choose the path, after that Eclipse restarts and this always looked awkward. Additionally. On the other side, Android Studio uses modules to manage and organize your code modules have their own Gradle build files which mean it can state their own dependencies. In compare AS looks more natural, but if you have been using Eclipse for some time, then it takes a little bit time to get used to.
- **System stability:** Eclipse is simply Java based software and a larger IDE in comparison with Android Studio, so it needs considerably higher amount of RAM space with a high CPU speed to function properly. Failure to meet this criterion causes Eclipse crashing and getting unresponsive. On the other hand, Android Studio is now released with very less

bugs, and provides a more stable performance guarantee than Eclipse and the system needs are lower too. AS is quick, while you need 1 or 2 minutes for building release versions of complex projects in Eclipse, but can make the same project within 30 seconds in AS.

- **Drag-and-Drop:** Android Studio has GUI (Graphical User Interface), but Eclipse does not have. However, the drag-and-drop feature is not essential for coders, who are not very much concerned regarding the visual elements of their applications. A developer needs to have detailed knowledge of Visual Basic, so that the developer can use the drag-and-drop feature appropriately. It's a new feature in Android Studio, but its state of being absent in Eclipse does not matter greatly.

Android Studio is certainly a step ahead of Eclipse, which lost its position in less than a year as the main IDE for android application development and became died out. Also, as Android Studio has been developed by android programmers, has much more compatibility with the operating system (OS) and specific tools. Official site: developer.android.com

4.3 Application development

4.3.1 Introduction to WBKAPP

After beginning with the programming of our app, we have to talk about the libraries (for more information about what a library is see chapter 2.3.3) that have been used to develop itself in a proper and easier way.

Bluetooth Low energy programming is not as developed over the years as classic bluetooth because has appeared few years ago with the IoT boom. Although communication with BLE device seems to be easy at first sight, it is more sophisticated as we have seen in other chapters because is based on services, characteristics and UUID, there are some points that can confuse developers. Processing only one read/write operation at a time, not reusing BluetoothGatt after disconnecting are the most important but implicit assumptions you should keep in mind. Despite this, grasping knowledge of BLE implementation is really valuable because it opens up new possibilities to create great products such as smart bulbs, armbands, home automation systems, etc. It is necessary to know how to manipulate those devices and BLE is a great way of doing it.

RxAndroidBLE is a powerful helper for Android's Bluetooth Low Energy headaches. It is backed by RxJava, implementing complicated APIs as handy reactive observables. The library does for you (Polidea 2016):

- Asynchronous operations support (read, write and notifications).
- Threading management in order to meet Android contracts.
- Connection and operation error handling.

The first point of the library is the one we are interested on, as if we do not use this library, the code just for the connection or discovery becomes more and more confusing and difficult as we can see in the official page of android developers original official sample. In this thesis, we are not going to discuss all library items as it is really difficult programming and is not our objective. We just limit to understand it and apply what is already done to help developers. Further code shown in following chapters

GraphView is a custom View that can be used to plot diagrams in your Android app. Because it extends the base View class of Android, you can use it like any other View in your layout. You can create simple graphs completely declarative in the layout XML file, or create more complex graphs programmatically in your Java code (Jonas Gehring 2017).

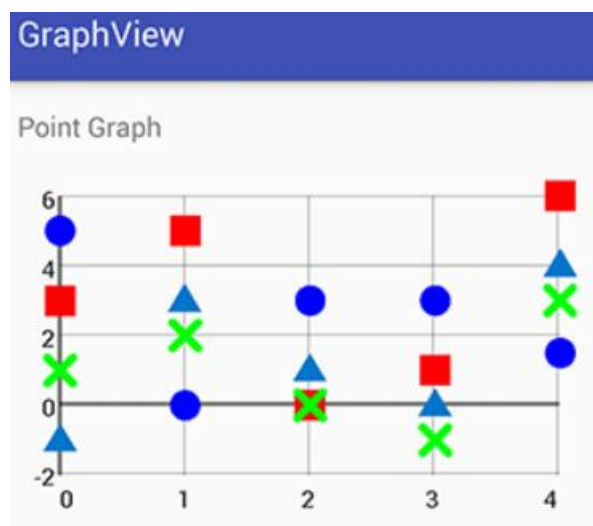


Figure 4.3: Point Graph example

We are going to use this library just to plot the data that are received from the Arduino through the bluetooth module to have a fast way to check the data received from the sensors and control the production of the machine. Key features supported:

- Different plotting types
- Draw multiple series of data
- Real-time / Live Chart
- Secondary Scale learn more
- Tap Listener
- Show legend
- Custom label formatter

- Handle incomplete data: It's possible to give the data in different frequency.
- Viewport
- Scrolling and Scaling / Zooming

4.3.2 Function flow in the application

When we want to begin developing our application, what we need to focus first is on the structure it is going to have and the functionality flow it is going to follow. On figure 4.4 a schematic flow is drawn:

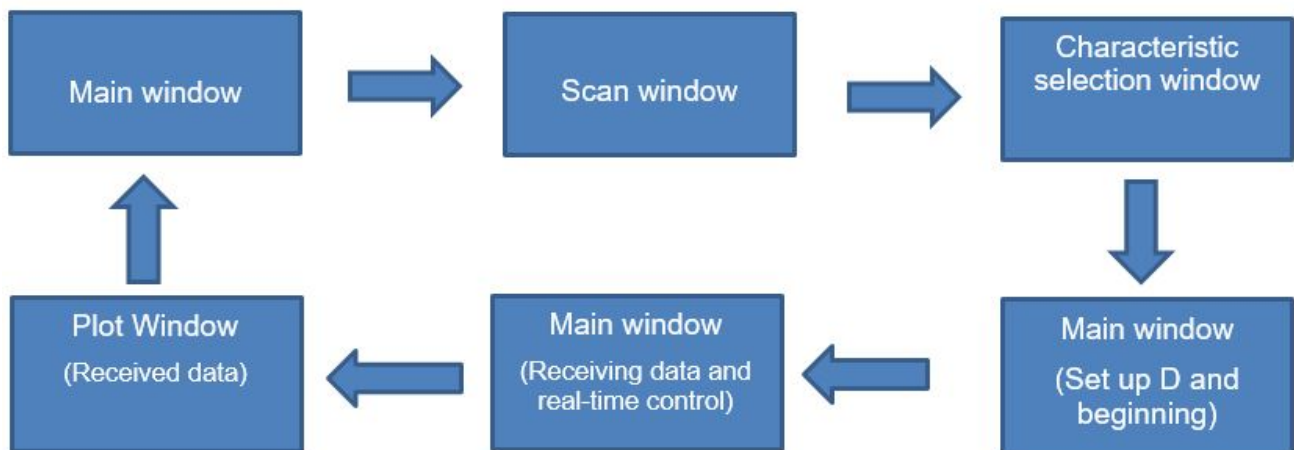


Figure 4.4: Application Flow scheme

To navigate through the different windows, we are going to use button elements as hardware native buttons that are in the bottom sides of the screen of Android smartphones as they are easy to program and so easy to understand to the final user. Putting into words the flow chart:

1. Start of the app, initialized in the main screen / window.
2. Push Scan Button.
3. Scan window initialize.
4. Push Start Scan Button.
5. Select device from the devices discovered in the list.
6. Initialize characteristic selection window.
7. Push Connect & Discover services button.
8. Select read / write / notify characteristic.
9. Auto initialize of the main window.
10. Set diameter, push register notification button.
11. Write 1 to begin.
12. Automatic start to receive data and show instant value

13. Push Plot button for start Plot window and shows a graph with data received.

14. End of cycle.

4.3.3 Main classes and methods

In this chapter, we are going to show how we can program in java to achieve the flow chart shown before, there are some methods that are necessary in all classes. The whole source code are posted in the Annex java classes and XML layouts (Graphical user interface code):

- `onCreate()`: this defines what the activity is going to do when it is initialized, in other words, what is going to show as we also have to initialize the XML code (buttons, textViews, etc.):

```
protected void onCreate(Bundle savedInstanceState) {  
    super.onCreate(savedInstanceState);  
    setContentView(R.layout.activity_plotforce); //charges the XML layout  
    GraphView graph = (GraphView) findViewById(R.id.graphf);  
    ...  
    ...  
}
```

- Initialize variables that are going to be used in the class, with each get and set methods that returns and change the value of each variable:

```
final static double k = 177;  
double diam;  
public int getDiam() {  
    return diam;  
}  
public void setDiam(int diam) {  
    this.diam = diam;  
}
```

- Set the package where the information of the app is saved and import the libraries we are going to use:

```
import android.widget.EditText;  
import android.widget.ListView;  
import android.widget.TextView;  
import android.widget.Toast;
```

- Button configuration to program what to do when a button is pushed. Example: setting up diameter:

```
public void onClick(View view) {  
    diam = Integer.valueOf(etdiam.getText().toString());  
    String s = String.valueOf(diam);
```

```

        tvdiam.setText(s);
        Toast.makeText(MainActivity.this, "Diameter saved : " + getDiam(),
Toast.LENGTH_SHORT).show();
        etdiam.setText("");
    }

});

```

- onRestart, onPause, onDestroy, onResume, etc. program what happen when that facts happen, for example, closing the app or pause because you let the app running in the background as you do other things with other apps in your smartphone:

```

@Override
public void onPause() {
    super.onPause();

    if (isScanning()) {
        /*
         * Stop scanning in onPause callback. You can use rxlifecycle for
convenience. Examples are provided later.
         */
        scanSubscription.unsubscribe();
    }
}

```

That are the basic methods that are implemented inside the application. For more complicated methods, see the following chapters where we discuss and show the main methods that make the application run. As said before in chapter 4.3.1, we use the BLE library RxAndroidBle, so some of the methods are already deep implemented on it and we are not able to describe in deep what are already doing as it is difficult to understand at our level.

4.3.3.1 ScanActivity class

In this class, what happen in the scanning window is implemented and some methods as showed following are the main ones, whole code Annex E:

- Setting up Start Scan button(scan_toggle_btn):

```

@OnClick(scan_toggle_btn)
public void onScanToggleClick() {

    if (isScanning()) {
        scanSubscription.unsubscribe();
    } else {
        scanSubscription = MainActivity.rxBleClient.scanBleDevices()
            .observeOn(AndroidSchedulers.mainThread())
            .doOnUnsubscribe(this::clearSubscriptionSCAN)
            .subscribe(resultsAdapter::addScanResult,
this::onScanFailure); //adds a device when it is available
    }
}

```

```

    updateButtonUIState();
}

```

- Handle when it is something wrong and tell how to solve it:

```

private void handleBleScanException(BleScanException bleScanException) {
    switch (bleScanException.getReason()) {
        case BleScanException.BLUETOOTH_NOT_AVAILABLE:
            Toast.makeText(ScanActivity.this, "Bluetooth is not available",
                Toast.LENGTH_SHORT).show();
            break;
        case BleScanException.BLUETOOTH_DISABLED:
            Toast.makeText(ScanActivity.this, "Enable bluetooth and try
                again", Toast.LENGTH_SHORT).show();
            break;
        case BleScanException.LOCATION_PERMISSION_MISSING:
            Toast.makeText(ScanActivity.this,
                "On Android 6.0 location permission is required. Implement
                Runtime Permissions", Toast.LENGTH_SHORT).show();
            break;
        case BleScanException.LOCATION_SERVICES_DISABLED:
            Toast.makeText(ScanActivity.this, "Location services needs to be
                enabled on Android 6.0", Toast.LENGTH_SHORT).show();
            break;
        case BleScanException.BLUETOOTH_CANNOT_START:
        default:
            Toast.makeText(ScanActivity.this, "Unable to start scanning",
                Toast.LENGTH_SHORT).show();
            break;
    }
}

```

- Configure the list where the results are going to be shown and setting up what happen when you select one device of the list:

```

private void configureResultList() {
    recyclerView.setHasFixedSize(true);
    LinearLayoutManager recyclerViewLayoutManager = new LinearLayoutManager(this);
    recyclerView.setLayoutManager(recyclerViewLayoutManager);
    resultsAdapter = new ScanResultsAdapter();
    recyclerView.setAdapter(resultsAdapter);
    resultsAdapter.setOnItemClickListener(view -> {
        final int childAdapterPosition =
            recyclerView.getChildAdapterPosition(view);
        final RxBleScanResult itemAtPosition =
            resultsAdapter.getItemAtPosition(childAdapterPosition);
        onAdapterItemClick(itemAtPosition);
    });
}

private boolean isScanning() {
    return scanSubscription != null;
}

```

```

    }

    private void onAdapterItemClick(RxBleScanResult scanResults) {
        final String macAddress = scanResults.getBleDevice().getMacAddress();
        final Intent intent = new Intent(this,
ServiceDiscoveryExampleActivity.class);
        intent.putExtra(ServiceDiscoveryExampleActivity.EXTRA_MAC_ADDRESS,
macAddress);
        startActivity(intent);
    }

```

4.3.3.2 ScanResultsAdapter class

This class is just an Adapter as the name says, what this means in other words is setting up an Adapter so we can use the ListView to show devices and define what we want to show. Whole code Annex F:

- Add a device which have been detected and avoiding to repeat devices:

```

void addScanResult(RxBleScanResult bleScanResult) {
    // Not the best way to ensure distinct devices, just for sake on the demo.

    for (int i = 0; i < data.size(); i++) {

        if (data.get(i).getBleDevice().equals(bleScanResult.getBleDevice())) {
            data.set(i, bleScanResult);
            notifyItemChanged(i);
            return;
        }
    }

    data.add(bleScanResult);
    Collections.sort(data, SORTING_COMPARATOR);
    notifyDataSetChanged();
}

```

- Setting up methods to set the fields as rssi and MAC address, to know the position of the device in the list to use it when it is pushed:

```

public RxBleScanResult getItemAtPosition(int childAdapterPosition) {
    return data.get(childAdapterPosition);
}

@Override
public int getItemCount() {
    return data.size();
}

@Override
public void onBindViewHolder(ViewHolder holder, int position) {
    final RxBleScanResult rxBleScanResult = data.get(position);
    final RxBleDevice bleDevice = rxBleScanResult.getBleDevice();
}

```

```

        holder.line1.setText(String.format("%s (%s)", bleDevice.getMacAddress(),
bleDevice.getName()));
        holder.line2.setText(String.format("RSSI: %d",
rxBleScanResult.getRssi()));
    }

    @Override
    public ViewHolder onCreateViewHolder(ViewGroup parent, int viewType) {
        final View itemView =
LayoutInflater.from(parent.getContext()).inflate(android.R.layout.two_line_list_it
em, parent, false);
        itemView.setOnClickListener(onClickListener);
        return new ViewHolder(itemView);
    }

    void setOnAdapterItemClickListener(OnAdapterItemClickListener
onAdapterItemClickListener) {
        this.onAdapterItemClickListener = onAdapterItemClickListener;
    }
}

```

4.3.3.3 ServiceDiscoveryExampleActivity class

In this class, it is shown what rules the characteristic selection window. The user has the responsibility of choosing the proper characteristic to run the app properly. Whole code Annex G.

- Setting up what happen when we push the button to discover the services and characteristics:

```

@OnClick(R.id.connect)
public void onConnectToggleClick() {
    bleDevice.establishConnection(false)
        .flatMap(RxBleConnection::discoverServices)
        .first() // Disconnect automatically after discovery
        .compose(bindUntilEvent(PAUSE))
        .observeOn(AndroidSchedulers.mainThread())
        .doOnUnsubscribe(this::updateUI)
        .subscribe(adapter::swapScanResult, this::onConnectionFailure);

    updateUI();
}

```

- Configure what happens when the services and characteristics are discovered and then added to the list; what happen when we select one:

```

private void configureResultList() {
    recyclerView.setHasFixedSize(true);
    LinearLayoutManager recyclerViewLayoutManager = new LinearLayoutManager(this);
    recyclerView.setLayoutManager(recyclerViewLayoutManager);
    adapter = new DiscoveryResultsAdapter();
    recyclerView.setAdapter(adapter);
    adapter.setOnItemClickListener(view -> {
        final int childAdapterPosition =
recyclerView.getChildAdapterPosition(view);
    });
}

```

```

        final DiscoveryResultsAdapter.AdapterItem itemAtPosition =
adapter.getItem(childAdapterPosition);
        onAdapterItemClick(itemAtPosition);
    });
}

private void onAdapterItemClick(DiscoveryResultsAdapter.AdapterItem item) {

    if (item.type == DiscoveryResultsAdapter.AdapterItem.CHARACTERISTIC) {
        MainActivity.characteristicUuid = item.uuid;
        MainActivity.macAddress = macAddress;
        final Intent intent = new Intent(this, MainActivity.class);
        startActivity(intent
            .addFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP |
Intent.FLAG_ACTIVITY_SINGLE_TOP));
        finish();
        startActivity(intent);
    } else {
        //noinspection ConstantConditions
        Snackbar.make(findViewById(android.R.id.content),
R.string.not_clickable, Snackbar.LENGTH_SHORT).show();
    }
}
}

```

- Check if it is connected properly and showing a message when there is something wrong with the connection:

```

private boolean isConnected() {
    if (bleDevice != null) {
        return bleDevice.getConnectionState() ==
RxBleConnection.RxBleConnectionState.CONNECTED;
    } else {
        return false;
    }
}

private void onConnectionFailure(Throwable throwable) {
    //noinspection ConstantConditions
    Snackbar.make(findViewById(android.R.id.content), "Connection error: " +
throwable, Snackbar.LENGTH_SHORT).show();
}

private void updateUI() {
    connectButton.setEnabled(!isConnected());
}
}

```

4.3.3.4 DeviceActivity class

This class has just been automatized so the user do not have to do nothing. Whole code in Annex H.

4.3.3.5 DiscoveryResultsAdapter class

This class do exactly the same as 4.3.3.2 ScanResultsAdapter class but, instead of showing devices in the list, the adapter show de services and characteristics. Code in Annex I.

- Detects the type of service and shows information to the user about that service's characteristics.

```
private String getServiceType(BluetoothGattService service) {
    return service.getType() == BluetoothGattService.SERVICE_TYPE_PRIMARY ?
    "primary" : "secondary";
}

private boolean isCharacteristicNotifiable(BluetoothGattCharacteristic
characteristic) {
    return (characteristic.getProperties() &
BluetoothGattCharacteristic.PROPERTY_NOTIFY) != 0;
}

private boolean isCharacteristicReadable(BluetoothGattCharacteristic
characteristic) {
    return ((characteristic.getProperties() &
BluetoothGattCharacteristic.PROPERTY_READ) != 0);
}

private boolean isCharacteristicWriteable(BluetoothGattCharacteristic
characteristic) {
    return (characteristic.getProperties() &
(BluetoothGattCharacteristic.PROPERTY_WRITE
| BluetoothGattCharacteristic.PROPERTY_WRITE_NO_RESPONSE)) != 0;
}
}
```

- Initialize the adapter item.

```
static class AdapterItem {

    static final int SERVICE = 1;
    static final int CHARACTERISTIC = 2;
    final int type;
    final String description;
    final UUID uuid;

    AdapterItem(int type, String description, UUID uuid) {
        this.type = type;
        this.description = description;
        this.uuid = uuid;
    }
}
```


4.3.3.6 MainActivity class

This is the class that makes the main screen run where the information to the user is shown and where the action buttons to initialize the scan window, start of receiving data, setting up diameter, etc. is done. Whole code in Annex J.

- onCreate(), where all information of the layout and button actions are implemented, is too large to post it here, go to Annex J.
- Setting up what happen when you click connect button:

```
@OnClick(R.id.connectBT)
public void ConnectBT() {
    if (macAddress!= null & characteristicUuid!=null) {
        if (isConnected()) {
            triggerDisconnect();
            updateUIscan();
        } else {
            connectionObservable.subscribe(rxBleConnection -> {
                Log.d(getClass().getSimpleName(), "Hey, connection has been
established!");
                runOnUiThread(this::updateUIscan);
            }, this::onConnectionFailure);
        }
    }
}
```

- Setting up what happen when you click write button:

```
@OnClick(R.id.writeBT)
public void onWriteClick() {
    if (isConnected()) {
        connectionObservable
            .flatMap(rxBleConnection ->
rxBleConnection.writeCharacteristic(characteristicUuid, getInputBytes()))
            .observeOn(AndroidSchedulers.mainThread())
            .subscribe(bytes -> {
                onWriteSuccess();
            }, this::onWriteFailure);
    }
}
```

- Setting up what happen when you click register notifications button:

```
@OnClick(R.id.notifyBT)
public void NotifyBT() {
    if (isConnected()) {
        connectionObservable
            .flatMap(rxBleConnection ->
rxBleConnection.setupNotification(characteristicUuid))
            .doOnNext(notificationObservable ->
runOnUiThread(this::notificationHasBeenSetUp))
            .flatMap(notificationObservable -> notificationObservable)
```

```

        .observeOn(AndroidSchedulers.mainThread())
        .subscribe(this::onNotificationReceived,
this::onNotificationSetupFailure);
    }
}

```

- What to do when a data is received, for information about the formula, see results chapter:

```

private void onNotificationReceived(byte[] bytes) {
    //noinspection ConstantConditions
    changelist.add(bytesToString(bytes));
    red = Double.valueOf(bytesToString(bytes));
    force = (red/k)*(diam/80)*1517;
    lecturaBT.setText(String.valueOf((force)));
}

```

4.3.3.7 Plotforce class

This class is just a call to the library and takes the data from the list that is in the main screen list, not the received one. As the code is just initialize the graph and plotting it, whole code is shown in Annex K.

4.3.3.8 SampleApplication class

This class is just the base of the library where the object `RxBleClient` is initialized as shown, whole code Annex M:

```

private RxBleClient rxBleClient;
public static RxBleClient getRxBleClient(Context context) {
    SampleApplication application = (SampleApplication)
context.getApplicationContext();
    return application.rxBleClient;
}

```

4.3.4 Arduino code

After developing our Android application, we face our next objective, use an Arduino to test the functionality of what we have done. For this, we have used the `SoftwareSerial` library, which is Arduino native.

The Arduino hardware has built-in support for serial communication on pins 0 and 1 (which also goes to the computer via the USB connection). The native serial support happens via a piece of hardware (built into the chip) called a UART. This hardware allows the Atmega chip to receive serial communication even while working on other tasks, as long as there room in the 64 byte serial buffer.

The `SoftwareSerial` library has been developed to allow serial communication on other digital pins of the Arduino, using software to replicate the functionality (hence the name "SoftwareSerial"). It is possible to have multiple software serial ports with speeds up to 115200

bps. A parameter enables inverted signaling for devices which require that protocol (Mikal Hart 2017).

The whole Arduino code is shown in Annex R. Main method (loop) is shown:

```
void loop() {  
  // run over and over  
  if (mySerial.available() && cont < 855) {  
    int i;  
    int k;  
    for (i = 0 ; i < 855; i++) {  
      k = myArray[i];  
      mySerial.print(k);  
      cont++;  
      Serial.print(cont); // To control by serial monitor what value is sending.  
      delay(100);  
    }  
  }  
  if (cont > 855) {  
    software_Reset();  
  }  
}
```

Where myArray[] is an array with the values of the test provided by my supervisor multiplied by 10000 to simplify the code (is better to send integer to float numbers). As there was so much values, up to 2200, I did a selection removing some values that were repeated so much times, ending in 855 values. When all values are sent, it reset itself.

5 Results

5.1 Validation of the functionality

To prove the functionality of our application, we are going to follow the steps described in chapter 4.3.2 showing it with screenshots of real smartphone device and an Arduino compiled with the code shown and a bluetooth module HM-10 connected to it as it indicates in Annex R. First of all, we made a modification from the formula provided by sensor calibration as:

$$F = \frac{UrD}{k * 80} * C \quad \text{Equation 5.1}$$

where F is the clamping force in N, Ur is the relative voltage, D is the diameter, k is the conversion factor kN to mV/V and equal to 117 (already multiplied by 10000 as the Ur sent through bluetooth is multiplied by 10000 also) and C is conversion factor kN to N (C = 1000).

In the first test, we realized that because of rounding errors because of program code, there was an error we had to fix as following. The calculation results we had were much lower than real ones (7180 N vs 10888 N), so we decided to introduce an error factor (S) calculated as:

$$S = \frac{10888}{7180} \quad \text{Equation 5.2}$$

$$Cm = 1000S \quad \text{Equation 5.3}$$

$$Fm = \frac{UrD}{k * 80} * Cm \quad \text{Equation 5.4}$$

where Fm is the adjusted clamping force and Cm is the modified conversion factor, which includes the error factor calculated in equation 5.2.

As we have demonstrated the equation change, we can continue with the validation of the functionality. Figure 5.1 shows the main window as it is created.

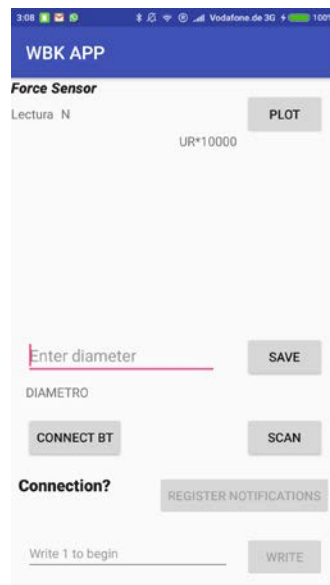


Figure 5.1 Main window clean

As we push scan button and Start Scan Button, the application starts to discover Low energy bluetooth devices as shown in Figure 5.2.



Figure 5.2 Scan window

When we click in our device from the list and in the next window we push de service discovery button, this is shown:

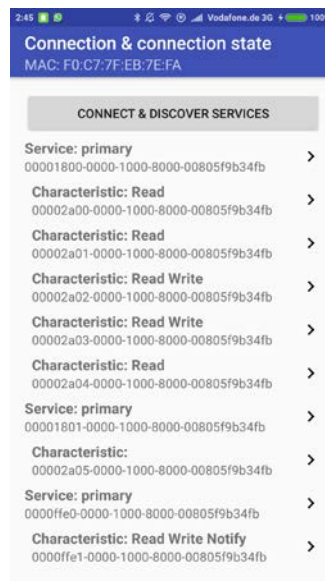


Figure 5.3 Service discovery window

As we can see, the characteristic that was expected in chapter 4.1.2 to appear is shown (Read, write, Notify), that is the one we must click on; when done and 1 is written in the text box ,main screen is shown as this receiving data from the module:



Figure 5.4: Received data in main screen

As last step, we have to check that the plotting window works. We compare the expected result done with matLab with our plot done with our application (taking into account that we deleted repeated values to storage them in the Arduino), figures 5.5 and 5.6:



Figure 5.5 Graph made with WBKAPP

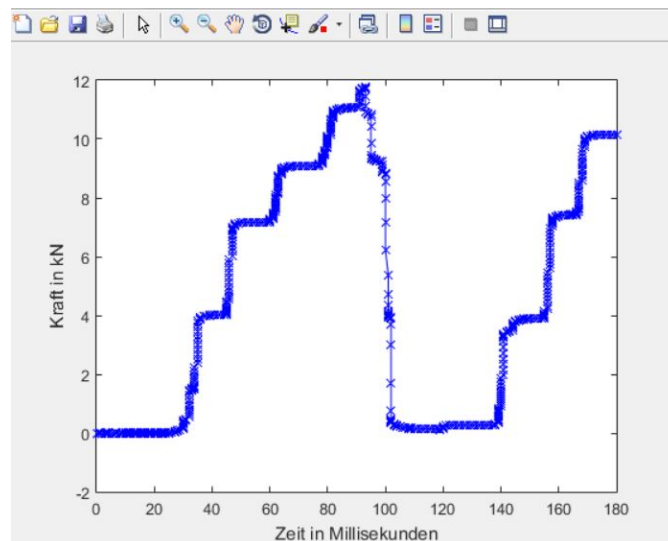


Figure 5.6 Graph made with MatLab

From this two figures, we know our application works plenty.

5.2 User's guide

Installation for the installation: Building an .apk file that is the compiled and installation prepared file is really easy in Android studio, you just have to click on Build and then Build APK and them, it compiles and there is no errors, the installation file is constructed.

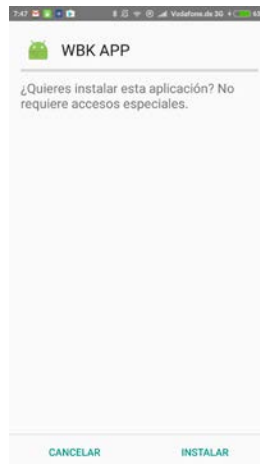


Figure 5.7: Executing .apk window (Spanish)

To install that file on a smartphone, is as easy as sending that .apk file to the device, by some kind of Cloud or e-mail and then execute it (must have external application installation on, this is done in developer options inside the smartphone); when executing it, figure 5.7 must be shown (or similar depending on the device).

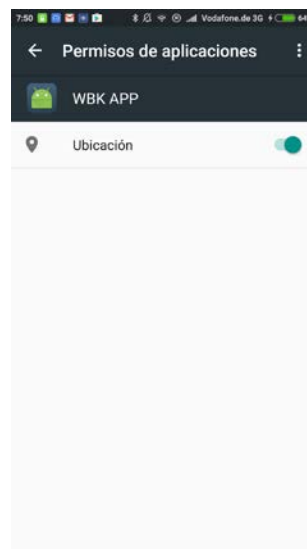


Figure 5.8 Location permissions (Spanish)

Due to location permissions are compulsory to use Bluetooth low energy in android, you must enable them as shown in figure 5.8.

As we based the application on the flow chart that we described on chapter 4.3.2, what we said there is also true at the end. To navigate through the different windows, we use button elements as hardware native buttons that are in the bottom sides of the screen of Android smartphones which are easy to understand to the final user. Steps to follow

1. Start of the app, initialized in the main screen / window.
2. Push Scan Button.
3. Scan window initialize.
4. Push Start Scan Button.

5. Select device from the devices discovered in the list.
6. Initialize characteristic selection window.
7. Push Connect & Discover services button.
8. Select read / write / notify characteristic.
9. Auto initialize of the main window.
10. Set diameter, push register notification button.
11. Write 1 to begin.
12. Automatic start to receive data and show instant value.
13. Push Plot button for start Plot window and shows a graph with data received.
14. End of cycle.

6 Assessment

In the term of bluetooth research, information have been easy to find as there is lot of information in that field and it is well studied. Even the official page of bluetooth ([bluetooth.com](https://www.bluetooth.com)) redirects you to Wikipedia, showing that it is a reliable information. There is also lot of bibliography.

In the OS target research, it has been easy to find studies that shows that Android is various steps ahead the second OS: iOS. Choosing a bluetooth module was also easy as there is lot of information about both modules selected as possibilities.

State of the art was the most difficult chapter as there is not so much information about this field in industry as it is beginning to be developed.

Developing the app was difficult until we found the RxAndroidBLE library and GraphView but, even with them, it has been a huge challenge and has to be polished (as the errors caused by approximations, some bugs in the app that obligate you to follow the steps mentioned), it is not user-friendly but efficient.

7 Summary & Outlook

7.1 Summary

This thesis has accomplished the objectives it has been created for, four different parts: bluetooth research, choose an OS target, choose a Bluetooth module and, finally, develop an Android app as we decided to do in the OS target selection. These objectives we had to cover have been reached.

Even though programming knowledge is really important to understand the main part of the thesis as the name indicates ("Development of a bluetooth application for measuring tension forces of a turning jaw"), programming principles as java basics to advanced libraries are shown.

There is also huge documentation about bluetooth research about both technologies: bluetooth classic and low energy bluetooth (BLE). Module selection based on technical data provided by datasheets is also done.

In this thesis, Annex are really important to understand the whole work.

7.2 Outlook

As a first view of the application, you can see this is a first idea of what is going to be developed in the future by wireless communication enterprises that have huge groups of programmers but, as a first idea, it do its job. A future implementation for industrial environment usage is the need of a studio about the error made by the programming approximations as it has been solved in this project with a non-professional but effective way.

Another fix for professional industrial implementation, is the use of professional micro-controllers and bluetooth modules as this ones are thought for DIY projects ("Do it yourself"), cleaning the code and having a professional graphical view because it is not user-friendly but it is efficient.

List of Figures

Figure 2.1 : Radio Frequency Spectrum (Wayne Staab 2013)	3
Figure 2.2: Bluetooth and WiFi Channels (Raajit Lall 2013)	4
Figure 2.3: Master – Slave diagram (Hung Bui 2016)	5
Figure 2.4: Bluetooth protocol stack	8
Figure 2.5: Server – Client Scheme	12
Figure 2.6: Android Studio GUI (Željko Plesac 2013)	15
Figure 2.7: XML design in Android Studio (Željko Plesac 2013)	19
Figure 3.1 Block diagram of wireless sensor system (Suprock, Nichols 2009)	21
Figure 3.2: Torque bridge on cutting tool with electrical diagram of gage bridge and schema (Suprock, Nichols 2009)	22
Figure 3.3: : Torque sensor integrated tool and transmitter (Suprock, Nichols 2009)	23
Figure 3.4: : Comparison between wireless communication tech. (Johen Schiler 2015)	24
Figure 3.5: Smart house example interface	25
Figure 3.6: ZigBee module	26
Figure 3.7: Block diagram for monitoring and controlling	27
Figure 3.8 : Temperature sensor	27
Figure 3.9: Gas sensor	28
Figure 3.10: Smartwatch GPS sensor	29
Figure 4.1: HC-05 Module	32
Figure 4.2 HM-10 Official vs fake	34
Figure 4.3: Point Graph example	39
Figure 4.4: Application Flow scheme	40
Figure 5.1 Main window clean	52
Figure 5.2 Scan window	52
Figure 5.3 Service discovery window	53
Figure 5.4: Received data in main screen	53
Figure 5.5 Graph made with WBKAPP	54
Figure 5.6 Graph made with MatLab	54
Figure 5.7: Executing .apk window (Spanish)	55
Figure 5.8 Location permissions (Spanish)	55

List of Tables

Table 2.1: Bluetooth classes (Bakker, McMichael Gilster 2002)	5
Table 4.1: Bluetooth classic vs BLE	30
Table 4.2 Specifications Summary	35
Table 4.3: Worldwide Smartphone Sales to End Users by Operating System in 4Q16, Thousands of Units (Gartner 2017)	35
Table 4.4 Android version market share 2016 (Android developers 2017)	36

References

- Android developers (2017): Dashboards. Platform Versions. Android developers. 2016. Available online at <https://developer.android.com/about/dashboards/index.html>, updated on 2017.
- Bakker, D. M.; McMichael Gilster, Diane (2002): Bluetooth end to end. [a practical guide for today's IT professionals ...]. New York, NY: M&T Books (End to end).
- Bluetooth developers (2016): What is Bluetooth, Bluetooth LE. Bluetooth 101. Available online at <https://en.wikipedia.org/wiki/Bluetooth>.
- Gartner, Inc. (2017): Fierce Battle Between Apple and Samsung to Hold the No. 1. Global Smartphone Ranking. Available online at <http://www.gartner.com/newsroom/id/3609817>.
- Gomez, Carles; Oller, Joaquim; Paradells, Josep (2012): Overview and Evaluation of Bluetooth Low Energy. An Emerging Low-Power Wireless Technology. In *Sensors* 12 (12), pp. 11734–11753. DOI: 10.3390/s120911734.
- Grant, Kevin; Haseman, Chris (2014): Beginning android programming. Develop and design / Kevin Grant and Chris Haseman. San Francisco, Calif.: Peachpit Press.
- Hoisington, Corinne (2015): Android boot camp for developers using Java, comprehensive. A guide to creating your first Android apps. Second edition. Australia: Cengage Learning.
- Huang, Albert S.; Rudolph, Larry (2007): Bluetooth Essentials for Programmers. Cambridge: Cambridge University Press.
- Hung Bui (2016): Bluetooth Smart and the Nordic's Softdevices Part 2 Connection. Available online at <https://devzone.nordicsemi.com/blogs/841/bluetooth-smart-and-the-nordics-softdevice-part-2/>.
- Jesse Russell (2012): Integrated development environment. With assistance of Ronald Cohn.
- Johen Schiler (2015): Other types of networks: Bluetooth, Zigbee, & NFC. Available online at <https://www.slideshare.net/DilumBandara/other-types-of-networks-2015-3>.
- Jonas Gehring (2017): GraphView. Open source graph plotting library for Android. Available online at <http://www.android-graphview.org/simple-graph/>.
- K.Pavani (2012): Implementation Of Can And Zigbee Networks Based Industrial Monitoring And Control Applications. In *International Journal of Engineering Research & Technology* 1 (5). Available online at <http://www.ijert.org/view-pdf/550/implementation-of-can-and-zigbee-networks-based-industrial-monitoring-and-control-applications>.
- Liu, Feipeng (2013): Android Native Development Kit cookbook. A step-by-step tutorial with more than 60 concise recipes on Android NDK development skills / Feipeng Liu. Birmingham: Packt Publishing Limited.
- Mikal Hart (2017): SoftwareSerial Library. Available online at <https://www.arduino.cc/en/Reference/softwareSerial>.
- Polidea (2016): RxAndroidBle. Polonia. Available online at <https://polidea.github.io/RxAndroidBle/>, updated on 2017.
- Raajit Lall (2013): Too Many Cooks in the 2.4 GHz Kitchen? National Instruments. Available online at <http://www.ni.com/newsletter/51685/en/>.

Suprock, Christopher A.; Nichols, Jeffery S. (2009): A low cost wireless high bandwidth transmitter for sensor-integrated metal cutting tools and process monitoring. In *IJMMS* 2 (4), p. 441. DOI: 10.1504/IJMMS.2009.027128.

Townsend, Kevin (2015): Getting started with Bluetooth low energy. Revised first edition, second release. Beijing: O'Reilly.

Wayne Staab (2013): Bluetooth 101 – Part II. Available online at <http://hearinghealthmatters.org/waynesworld/2013/bluetooth-101-part-ii/>.

Yener, Murat; Dunder, Onur (2016): Expert Android® Studio. Indianapolis, IN: Wrox, a Wiley brand.

Željko Plesac (2013): Android Studio vs. Eclipse. Available online at <https://infinum.co/the-capsized-eight/android-studio-vs-eclipse-1-0>.

Appendix

ANNEX A


Guangzhou HC Information Technology Co.,Ltd.

Product Data Sheet

Module Data Sheet

Rev 1

1. 0	1.01						
2010/5/15	2011/4/6						

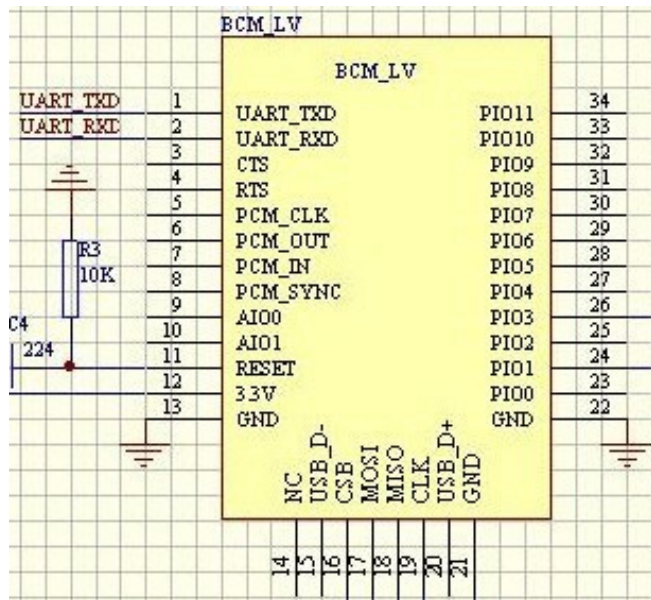
DRAWN BY :	Ling Xin		MODEL : HC-05	
CHECKED BY :	Eric Huang		Description: BC04 has external 8M Flash and EDR mode HC-05 is industrial, and compatible with civil HC-05	
APPD. BY:	Simon 		REV : 2.0	PAGES:
Former version introduction	Lingxin is the former of Wavesen.			

2. Features

- ☐ Wireless transceiver
- ☐ Sensitivity (Bit error rate) can reach -80dBm.
- ☐ The change range of output's power: -4 - +6dBm.
- ☐ Function description (perfect Bluetooth solution)
- ☐ Has an EDR module; and the change range of modulation depth: 2Mbps - 3Mbps.
- ☐ Has a build-in 2.4GHz antenna; user needn't test antenna.

- ☐ Has the external 8Mbit FLASH
- ☐ Can work at the low voltage (3.1V~4.2V). The current in pairing is in the range of 30~40mA. The current in communication is 8mA.
- ☐ PIO control can be switched.
- ☐ Has the standard HCI Port (UART or USB)
- ☐ The USB protocol is Full Speed USB1.1, and compliant with 2.0.
- ☐ This module can be used in the SMD.
- ☐ It's made through RoHS process.
- ☐ The board PIN is half hole size.
- ☐ Has a 2.4GHz digital wireless transceiver.
- ☐ Bases at CSR BC04 Bluetooth technology.
- ☐ Has the function of adaptive frequency hopping.
- ☐ Small (27mm×13mm×2mm).
- ☐ Peripheral circuit is simple.
- ☐ It's at the Bluetooth class 2 power level.
- ☐ Storage temperature range: -40 °C - 85°C, operating temperature range: -25 °C - +75°C
- ☐ Any wave inter Interference: 2.4MHz, the power of emitting: 3 dBm.
- ☐ Bit error rate: 0. Only the signal decays at the transmission link, bit error may be produced. For example, when RS232 or TTL is being processed, some signals may decay.
- ☐ Low power consumption
- ☐ Has high-performance wireless transceiver system
- ☐ Low Cost
- ☐ Application fields:
- ☐ Bluetooth Car Handsfree Device
- ☐ Bluetooth GPS

- ☐ Bluetooth PCMCIA , USB Dongle
- ☐ Bluetooth Data Transfer
- ☐ Software
- ☐ CSR



ANNEX B

HC Serial Bluetooth Products User Instructional Manual

1 Introduction

HC serial Bluetooth products consist of Bluetooth serial interface module and Bluetooth adapter, such as:

(1) Bluetooth serial interface module:

Industrial level: HC-03, HC-04(HC-04-M, HC-04-S)

Civil level: HC-05, HC-06(HC-06-M, HC-06-S)
HC-05-D, HC-06-D (with baseboard, for test and evaluation)

(2) Bluetooth adapter:

HC-M4

HC-M6

This document mainly introduces Bluetooth serial module. Bluetooth serial module is used for converting serial port to Bluetooth. These modules have two modes: master and slaver device. The device named after even number is defined to be master or slaver when out of factory and can't be changed to the other mode. But for the device named after odd number, users can set the work mode (master or slaver) of the device by AT commands.

HC-04 specifically includes:

Master device: HC-04-M, M=master

Slave device: HC-04-S, S=slaver

The default situation of HC-04 is slave mode. If you need master mode, please state it clearly or place an order for HC-04-M directly. The naming rule of HC-06 is same.

When HC-03 and HC-05 are out of factory, one part of parameters are set for activating the device. The work mode is not set, since user can set the mode of HC-03, HC-05 as they want.

The main function of Bluetooth serial module is replacing the serial port line, such as:

1. There are two MCUs want to communicate with each other. One connects to Bluetooth master device while the other one connects to slave device. Their connection can be built once the pair is made. This Bluetooth connection is equivalently liked to a serial port line connection including RXD, TXD

signals. And they can use the Bluetooth serial module to communicate with each other.

2. When MCU has Bluetooth slave module, it can communicate with Bluetooth adapter of computers and smart phones. Then there is a virtual communicable serial port line between MCU and computer or smart phone.

3. The Bluetooth devices in the market mostly are slave devices, such as Bluetooth printer, Bluetooth GPS. So, we can use master module to make pair and communicate with them.

Bluetooth Serial module's operation doesn't need drive, and can communicate with the other Bluetooth device who has the serial. But communication between two Bluetooth modules requires at least two conditions:

(1) The communication must be between master and slave.

(2) The password must be correct.

However, the two conditions are not sufficient conditions. There are also some other conditions basing on different device model. Detailed information is provided in the following chapters.

In the following chapters, we will repeatedly refer to Linvor's (Formerly known as Guangzhou HC Information Technology Co., Ltd.) material and photos.

2 Selection of the Module

The Bluetooth serial module named even number is compatible with each other; The slave module is also compatible with each other. In other word, the function of HC-04 and HC-06, HC-03 and HC-05 are mutually compatible with each other. HC-04 and HC-06 are former version that user can't reset the work mode (master or slave). And only a few AT commands and functions can be used, like reset the name of Bluetooth (only the slaver), reset the password, reset the baud rate and check the version number. The command set of HC-03 and HC-05 are more flexible than HC-04 and HC-06's. Generally, the Bluetooth of HC-03/HC-05 is recommended for the user.

Here are the main factory parameters of HC-05 and HC-06. Pay attention to the differences:

HC-05	HC-06
Master and slave mode can be switched	Master and slave mode can't be switched
Bluetooth name: HC-05	Bluetooth name: linvor
Password:1234	Password:1234

<p>Master role: have no function to remember the last paired slave device. It can be made paired to any slave device. In other words, just set AT+CMODE=1 when out of factory. If you want HC-05 to remember the last paired slave device address like HC-06, you can set AT+CMODE=0 after paired with the other device. Please refer the command set of HC-05 for the details.</p>	<p>Master role: have paired memory to remember last slave device and only make pair with that device unless KEY (PIN26) is triggered by high level. The default connected PIN26 is low level.</p>
<p>Pairing: The master device can not only make pair with the specified Bluetooth address, like cell-phone, computer adapter, slave device, but also can search and make pair with the slave device automatically.</p> <p>Typical method: On some specific conditions, master device and slave device can make pair with each other automatically. (This is the default method.)</p>	<p>Pairing: Master device search and make pair with the slave device automatically.</p> <p>Typical method: On some specific conditions, master and slave device can make pair with each other automatically.</p>
<p>Multi-device communication: There is only point to point communication for modules, but the adapter can communicate with multi-modules.</p>	<p>Multi-device communication: There is only point to point communication for modules, but the adapter can communicate with multi-modules.</p>
<p>AT Mode 1: After power on, it can enter the AT mode by triggering PIN34 with high level. Then the baud rate for setting AT command is equal to the baud rate in communication, for example: 9600.</p> <p>AT mode 2: First set the PIN34 as high level, or while on powering the module set the PIN34 to be high level, the Baud rate used here is 38400 bps.</p> <p>Notice: All AT commands can be operated only</p>	<p>AT Mode: Before paired, it is at the AT mode. After paired it's at transparent communication.</p>

when the PIN34 is at high level. Only part of the AT commands can be used if PIN34 doesn't keep the high level after entering to the AT mode. Through this kind of designing, set permissions for the module is left to the user's external control circuit, that makes the application of HC-05 is very flexible.	
During the process of communication, the module can enter to AT mode by setting PIN34 to be high level. By releasing PIN34, the module can go back to communication mode in which user can inquire some information dynamically. For example, to inquire the pairing is finished or not.	During the communication mode, the module can't enter to the AT mode.
Default communication baud rate: 9600, 4800-1.3M are settable.	Default communication baud rate: 9600, 1200-1.3M are settable.
KEY: PIN34, for entering to the AT mode.	KEY: PIN26, for master abandons memory.
LED1: PIN31, indicator of Bluetooth mode. Slow flicker (1Hz) represents entering to the AT mode2, while fast flicker(2Hz) represents entering to the AT mode1 or during the communication pairing. Double flicker per second represents pairing is finished, the module is communicable. LED2: PIN32, before pairing is at low level, after the pairing is at high level. The using method of master and slaver's indicator is the same. Notice: The PIN of LED1 and LED2 are connected with LED+.	LED: The flicker frequency of slave device is 102ms. If master device already has the memory of slave device, the flicker frequency during the pairing is 110ms/s. If not, or master has emptied the memory, then the flicker frequency is 750m/s. After pairing, no matter it's a master or slave device, the LED PIN is at high level. Notice: The LED PIN connects to LED+ PIN.
Consumption: During the pairing, the current is	Consumption: During the pairing, the current is

fluctuant in the range of 30-40mA. The mean current is about 25mA. After paring, no matter processing communication or not, the current is 8mA. There is no sleep mode. This parameter is same for all the Bluetooth modules.	fluctuant in the range of 30-40 m. The mean current is about 25mA. After paring, no matter processing communication or not, the current is 8mA. There is no sleep mode. This parameter is same for all the Bluetooth modules.
Reset: PIN11, active if it's input low level. It can be suspended in using.	Reset: PIN11, active if it's input low level. It can be suspended in using.
Level: Civil	Level: Civil

The table above that includes main parameters of two serial modules is a reference for user selection.

HC-03/HC-05 serial product is recommended.

3. Information of Package

The PIN definitions of HC-03, HC-04, HC-05 and HC-06 are kind of different, but the package size is the same: 28mm * 15mm * 2.35mm.

The following figure 1 is a picture of HC-06 and its main PINs. Figure 2 is a picture of HC-05 and its main PINs. Figure 3 is a comparative picture with one coin. Figure 4 is their package size information. When user designs the circuit, you can visit the website of Guangzhou HC Information Technology Co., Ltd. (www.wavesen.com) to download the package library of protle version.

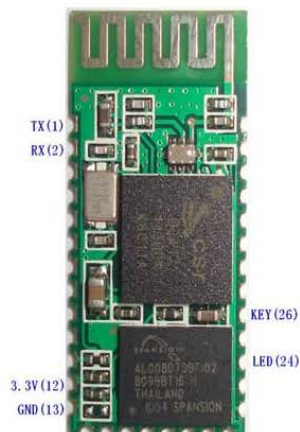


Figure 1 HC-06

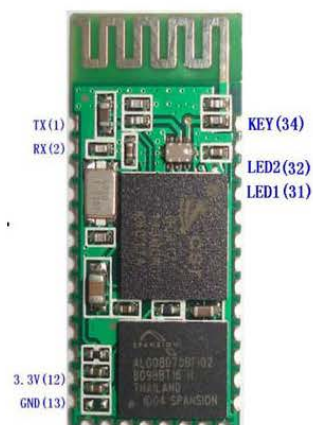


Figure 2 HC-05

ANNEX C

HM Bluetooth module datasheet

**The most complete, most convenient, the most stable of
Bluetooth data transmission, remote control, PIO
acquisition module**

---- Master and slave role in one

---- Remote control without other MCU

---- The PIO data acquisition without other MCU

13. Product parameters

BT Version: Bluetooth Specification V4.0 BLE

Send and receive no bytes limit.

Working frequency: 2.4GHz ISM band

Modulation method: GFSK(Gaussian Frequency Shift Keying)

RF Power: -23dbm, -6dbm, 0dbm, 6dbm, can modify through AT
Command AT+POWE.

Speed: Asynchronous: 6K Bytes

Synchronous: 6K Bytes

Security: Authentication and encryption

Service: Central & Peripheral UUID FFE0,FFE1

Power: +3.3VDC 50mA

Long range: Open space have 100 Meters with iPhone4s

Power: In sleep mode 400uA~1.5mA, Active mode 8.5mA.

Working temperature: -5 ~ +65 Centigrade

Size: HM-10 26.9mm x 13mm x 2.2 mm; HM-11 18*13.5*2.2mm

-----Last Version V524 2014-03-08 4

2. Product overview

Thanks for you choose our products. If you want to know more, www.inhuamao.cn can help you (Videos, New version datasheet, Module work flow, project Codes, etc.)

HM Bluetooth module use CSR BlueCore or TI CC2540, Master and slave roles in one, transmission version and remote control version and PIO state acquisition functions in one, Support the AT command modify module parameters, Convenient and flexible.

Transmission version can be used to transmit data between two Bluetooth devices.

Remote Control version can be used to Control PIO ports output high or low level without any other MCU.

The PIO state acquisition version can be used to acquisition PIO ports state without any other MUC. (Only support Bluetooth V2.1)

HM-01, HM-02, HM-03, HM-04, HM-05, HM-06, HM-07, HM-08, HM-09 is Bluetooth V2.1 version. Use CSR Chip.

HM-10, HM-11, HM-12 is Bluetooth V4.0 BLE version. Use TI Chip.

HM-01, HM-02, HM-09, HM-10 have same size and same pins.

HM-05, HM-06, HM-07, HM-11 have same size and same pins.

3. Product model

Models	VDD	Size(mm)	Flash	Chip	BT Version
HM-01	3.3V	26.9*13*2.2	8M	BC417143	V2.1+EDR
HM-02	2.5-3.7V	26.9*13*2.2	6M	BC3/BC4	V2.1
HM-03	2.5-3.7V	27.4*12.5*4.3	6M	BC3/BC4	V2.1
HM-04	3.3V	Not for sale			
HM-05	2.5-3.7V	13.5*18.5*2.3	6M	BC3/BC4	V2.1
HM-06	2.5-3.7V	13.5*18.5*2.3	6M	BC3/BC4	V2.1
HM-07	2.5-3.7V	13.5*18.5*2.3	8M		V2.1+EDR
HM-08	3.3V	26.9*13*2.5	8M	Class 1	V2.1+EDR
HM-09	2.5-3.7V	26.9*13*2.2	8M		V2.1+EDR
HM-10	2-3.7V	26.9*13*2.2	256Kb	CC2540/1	V4.0 BLE
HM-11	2.5-3.7V	13.5*18.5*2.2	256Kb	CC2540/1	V4.0 BLE
HM-15	5V	65*32*16	256KB	CC2540	V4.0 BLE

ANNEX D



CC2541

www.ti.com

SWRS110D – JANUARY 2012 – REVISED JUNE 2013

2.4-GHz Bluetooth™ low energy and Proprietary System-on-Chip

Check for Samples: [CC2541](#)

FEATURES

- **RF**
 - 2.4-GHz Bluetooth low energy Compliant and Proprietary RF System-on-Chip
 - Supports 250-kbps, 500-kbps, 1-Mbps, 2-Mbps Data Rates
 - Excellent Link Budget, Enabling Long-Range Applications Without External Front End
 - Programmable Output Power up to 0 dBm
 - Excellent Receiver Sensitivity (–94 dBm at 1 Mbps), Selectivity, and Blocking Performance
 - Suitable for Systems Targeting Compliance With Worldwide Radio Frequency Regulations: ETSI EN 300 328 and EN 300 440 Class 2 (Europe), FCC CFR47 Part 15 (US), and ARIB STD-T66 (Japan)
- **Layout**
 - Few External Components
 - Reference Design Provided
 - 6-mm × 6-mm QFN-40 Package
 - Pin-Compatible With CC2540 (When Not Using USB or I²C)
- **Low Power**
 - Active-Mode RX Down to: 17.9 mA
 - Active-Mode TX (0 dBm): 18.2 mA
 - Power Mode 1 (4-μs Wake-Up): 270 μA
 - Power Mode 2 (Sleep Timer On): 1 μA
 - Power Mode 3 (External Interrupts): 0.5 μA
 - Wide Supply-Voltage Range (2 V–3.6 V)
- **TPS62730 Compatible Low Power in Active Mode**
 - RX Down to: 14.7 mA (3-V supply)
 - TX (0 dBm): 14.3 mA (3-V supply)
- **Microcontroller**
 - High-Performance and Low-Power 8051 Microcontroller Core With Code Prefetch
 - In-System-Programmable Flash, 128- or 256-KB
 - 8-KB RAM With Retention in All Power Modes
 - Hardware Debug Support
 - Extensive Baseband Automation, Including Auto-Acknowledgment and Address Decoding
 - Retention of All Relevant Registers in All Power Modes
- **Peripherals**
 - Powerful Five-Channel DMA
 - General-Purpose Timers (One 16-Bit, Two 8-Bit)
 - IR Generation Circuitry
 - 32-kHz Sleep Timer With Capture
 - Accurate Digital RSSI Support
 - Battery Monitor and Temperature Sensor
 - 12-Bit ADC With Eight Channels and Configurable Resolution
 - AES Security Coprocessor
 - Two Powerful USARTs With Support for Several Serial Protocols
 - 23 General-Purpose I/O Pins (21 × 4 mA, 2 × 20 mA)
 - I²C interface
 - 2 I/O Pins Have LED Driving Capabilities
 - Watchdog Timer
 - Integrated High-Performance Comparator
- **Development Tools**
 - CC2541 Evaluation Module Kit (CC2541EMK)
 - CC2541 Mini Development Kit (CC2541DK-MINI)
 - SmartRF™ Software
 - IAR Embedded Workbench™ Available



Please be aware that an important notice concerning availability, standard warranty, and use in critical applications of Texas Instruments semiconductor products and disclaimers thereto appears at the end of this data sheet.

Bluetooth is a trademark of Bluetooth SIG, Inc..

ZigBee is a registered trademark of ZigBee Alliance.

PRODUCTION DATA information is current as of publication date. Products conform to specifications per the terms of the Texas Instruments standard warranty. Production processing does not necessarily include testing of all parameters.

Copyright © 2012–2013, Texas Instruments Incorporated

CC2541



SWRS110D – JANUARY 2012 – REVISED JUNE 2013

www.ti.com

SOFTWARE FEATURES

- **Bluetooth v4.0 Compliant Protocol Stack for Single-Mode BLE Solution**
 - Complete Power-Optimized Stack, Including Controller and Host
 - GAP – Central, Peripheral, Observer, or Broadcaster (Including Combination Roles)
 - ATT / GATT – Client and Server
 - SMP – AES-128 Encryption and Decryption
 - L2CAP
 - Sample Applications and Profiles
 - Generic Applications for GAP Central and Peripheral Roles
 - Proximity, Accelerometer, Simple Keys, and Battery GATT Services
 - More Applications Supported in [BLE Software Stack](#)
 - Multiple Configuration Options
 - Single-Chip Configuration, Allowing Applications to Run on CC2541
 - Network Processor Interface for Applications Running on an External Microcontroller
 - BTool – Windows PC Application for Evaluation, Development, and Test

APPLICATIONS

- 2.4-GHz *Bluetooth* low energy Systems
- Proprietary 2.4-GHz Systems
- Human-Interface Devices (Keyboard, Mouse, Remote Control)
- Sports and Leisure Equipment
- Mobile Phone Accessories
- Consumer Electronics

CC2541 WITH TPS62730

- **TPS62730** is a 2-MHz Step-Down Converter With Bypass Mode
- Extends Battery Lifetime by up to 20%
- Reduced Current in All Active Modes
- 30-nA Bypass Mode Current to Support Low-Power Modes
- RF Performance Unchanged
- Small Package Allows for Small Solution Size
- CC2541 Controllable

DESCRIPTION

The CC2541 is a power-optimized true system-on-chip (SoC) solution for both *Bluetooth* low energy and proprietary 2.4-GHz applications. It enables robust network nodes to be built with low total bill-of-material costs. The CC2541 combines the excellent performance of a leading RF transceiver with an industry-standard enhanced 8051 MCU, in-system programmable flash memory, 8-KB RAM, and many other powerful supporting features and peripherals. The CC2541 is highly suited for systems where ultralow power consumption is required. This is specified by various operating modes. Short transition times between operating modes further enable low power consumption.

The CC2541 is pin-compatible with the CC2540 in the 6-mm × 6-mm QFN40 package, if the USB is not used on the CC2540 and the I²C/extra I/O is not used on the CC2541. Compared to the CC2540, the CC2541 provides lower RF current consumption. The CC2541 does not have the USB interface of the CC2540, and provides lower maximum output power in TX mode. The CC2541 also adds a HW I²C interface.

The CC2541 is pin-compatible with the CC2533 RF4CE-optimized IEEE 802.15.4 SoC.

The CC2541 comes in two different versions: CC2541F128/F256, with 128 KB and 256 KB of flash memory, respectively.

For the CC2541 block diagram, see [Figure 1](#).

ANNEX E

```
package com.polidea.rxandroidble.sample;

import android.content.Intent;
import android.os.Bundle;
import android.support.v7.app.AppCompatActivity;
import android.support.v7.widget.LinearLayoutManager;
import android.support.v7.widget.RecyclerView;
import android.widget.Button;
import android.widget.Toast;

import com.polidea.rxandroidble.RxBleScanResult;
import com.polidea.rxandroidble.exceptions.BleScanException;

import butterknife.BindView;
import butterknife.ButterKnife;
import butterknife.OnClick;
import rx.Subscription;
import rx.android.schedulers.AndroidSchedulers;

import static com.polidea.rxandroidble.sample.R.id.scan_toggle_btn;

public class ScanActivity extends AppCompatActivity {

    @BindView(scan_toggle_btn)
    Button scanToggleButton;
    @BindView(R.id.scan_results)
    RecyclerView recyclerView;
    private Subscription scanSubscription;
    private ScanResultsAdapter resultsAdapter;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_example1);
        scanToggleButton = (Button) findViewById(scan_toggle_btn);
        recyclerView = (RecyclerView) findViewById(R.id.scan_results);
        ButterKnife.bind(this); // Annotate fields with @BindView and a view ID for
        // ButterKnife to find and automatically cast the corresponding view in your layout.
        configureResultList();
        onScanToggleClick();
    }

    @OnClick(scan_toggle_btn)
    public void onScanToggleClick() {

        if (isScanning()) {
            scanSubscription.unsubscribe();
        } else {
            scanSubscription = MainActivity.rxBleClient.scanBleDevices()
                .observeOn(AndroidSchedulers.mainThread())
                .doOnUnsubscribe(this::clearSubscriptionSCAN)
        }
    }
}
```

```
        .subscribe(resultsAdapter::addScanResult,
this::onScanFailure);//adds a device when it is available
    }

    updateButtonUIState();
}

private void handleBleScanException(BleScanException bleScanException) {

    switch (bleScanException.getReason()) {
        case BleScanException.BLUETOOTH_NOT_AVAILABLE:
            Toast.makeText(ScanActivity.this, "Bluetooth is not available",
Toast.LENGTH_SHORT).show();
            break;
        case BleScanException.BLUETOOTH_DISABLED:
            Toast.makeText(ScanActivity.this, "Enable bluetooth and try
again", Toast.LENGTH_SHORT).show();
            break;
        case BleScanException.LOCATION_PERMISSION_MISSING:
            Toast.makeText(ScanActivity.this,
"On Android 6.0 location permission is required. Implement
Runtime Permissions", Toast.LENGTH_SHORT).show();
            break;
        case BleScanException.LOCATION_SERVICES_DISABLED:
            Toast.makeText(ScanActivity.this, "Location services needs to be
enabled on Android 6.0", Toast.LENGTH_SHORT).show();
            break;
        case BleScanException.BLUETOOTH_CANNOT_START:
        default:
            Toast.makeText(ScanActivity.this, "Unable to start scanning",
Toast.LENGTH_SHORT).show();
            break;
    }
}

@Override
public void onPause() {
    super.onPause();

    if (isScanning()) {
        /*
         * Stop scanning in onPause callback. You can use rxlifecycle for
convenience. Examples are provided later.
         */
        scanSubscription.unsubscribe();
    }
}

private void configureResultList() {
    recyclerView.setHasFixedSize(true);
    LinearLayoutManager recyclerViewLayoutManager = new LinearLayoutManager(this);
    recyclerView.setLayoutManager(recyclerViewLayoutManager);
    resultsAdapter = new ScanResultsAdapter();
    recyclerView.setAdapter(resultsAdapter);
    resultsAdapter.setOnItemClickListener(view -> {
        final int childAdapterPosition =
recyclerView.getChildAdapterPosition(view);
```

```
        final RxBleScanResult itemAtPosition =
resultsAdapter.getItemAtPosition(childAdapterPosition);
        onAdapterItemClick(itemAtPosition);
    });
}

private boolean isScanning() {
    return scanSubscription != null;
}

private void onAdapterItemClick(RxBleScanResult scanResults) {
    final String macAddress = scanResults.getBleDevice().getMacAddress();
    final Intent intent = new Intent(this,
ServiceDiscoveryExampleActivity.class);
    intent.putExtra(ServiceDiscoveryExampleActivity.EXTRA_MAC_ADDRESS,
macAddress);
    startActivity(intent); // Automatic initialize of next window,
characteristics one
}

private void onScanFailure(Throwable throwable) {

    if (throwable instanceof BleScanException) {
        handleBleScanException((BleScanException) throwable);
    }
}

protected void clearSubscriptionSCAN() {
    scanSubscription = null;
    resultsAdapter.clearScanResults();
    updateButtonUIState();
}
// Changes the button text while it is scanning.
private void updateButtonUIState() {
    scanToggleButton.setText(isScanning() ? R.string.stop_scan :
R.string.start_scan);
}
}
```

ANNEX F

```
package com.polidea.rxandroidble.sample;

import android.support.v7.widget.RecyclerView;
```



```
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.TextView;
import com.polidea.rxandroidble.RxBleDevice;
import com.polidea.rxandroidble.RxBleScanResult;
import java.util.ArrayList;
import java.util.Collections;
import java.util.Comparator;
import java.util.List;

import butterknife.BindView;
import butterknife.ButterKnife;

class ScanResultsAdapter extends
RecyclerView.Adapter<ScanResultsAdapter.ViewHolder> {

    static class ViewHolder extends RecyclerView.ViewHolder {

        @BindView(android.R.id.text1)
        TextView line1;
        @BindView(android.R.id.text2)
        TextView line2;

        ViewHolder(View itemView) {
            super(itemView);
            ButterKnife.bind(this, itemView);
        }
    }

    interface OnAdapterItemClickListener {

        void onAdapterViewClick(View view);

    }

    private static final Comparator<RxBleScanResult> SORTING_COMPARATOR = (lhs,
rhs) -> {
    return
lhs.getBleDevice().getMacAddress().compareTo(rhs.getBleDevice().getMacAddress());
};
    private final List<RxBleScanResult> data = new ArrayList<>();
    private OnAdapterItemClickListener onAdapterItemClickListener;
    private final View.OnClickListener onClickListener = new
View.OnClickListener() {
        @Override
        public void onClick(View v) {

            if (onAdapterItemClickListener != null) {
                onAdapterItemClickListener.onAdapterViewClick(v);
            }
        }
    };

    void addScanResult(RxBleScanResult bleScanResult) {
        // Not the best way to ensure distinct devices, just for sake on the demo.

        for (int i = 0; i < data.size(); i++) {
```

```
        if (data.get(i).getBleDevice().equals(bleScanResult.getBleDevice())) {
            data.set(i, bleScanResult);
            notifyItemChanged(i);
            return;
        }
    }
    data.add(bleScanResult);
    Collections.sort(data, SORTING_COMPARATOR);
    notifyDataSetChanged();
}

void clearScanResults() {
    data.clear();
    notifyDataSetChanged();
}

public RxBleScanResult getItemAtPosition(int childAdapterPosition) {
    return data.get(childAdapterPosition);
}

@Override
public int getItemCount() {
    return data.size();
}

@Override
public void onBindViewHolder(ViewHolder holder, int position) {
    final RxBleScanResult rxBleScanResult = data.get(position);
    final RxBleDevice bleDevice = rxBleScanResult.getBleDevice();
    holder.line1.setText(String.format("%s (%s)", bleDevice.getMacAddress(),
bleDevice.getName()));
    holder.line2.setText(String.format("RSSI: %d",
rxBleScanResult.getRssi()));
}

@Override
public ViewHolder onCreateViewHolder(ViewGroup parent, int viewType) {
    final View itemView =
LayoutInflater.from(parent.getContext()).inflate(android.R.layout.two_line_list_it
em, parent, false);
    itemView.setOnClickListener(onClickListener);
    return new ViewHolder(itemView);
}

void setOnAdapterItemClickListener(OnAdapterItemClickListener
onAdapterItemClickListener) {
    this.onAdapterItemClickListener = onAdapterItemClickListener;
}
}
```

ANNEX G

```
package com.polidea.rxandroidble.sample;

import android.content.Intent;
import android.os.Bundle;
import android.support.design.widget.Snackbar;
import android.support.v7.widget.LinearLayoutManager;
```

```
import android.support.v7.widget.RecyclerView;
import android.widget.Button;
import com.polidea.rxandroidble.RxBleConnection;
import com.polidea.rxandroidble.RxBleDevice;
import com.trello.rxlifecycle.components.support.RxAppCompatActivity;
import butterknife.BindView;
import butterknife.ButterKnife;
import butterknife.OnClick;
import rx.android.schedulers.AndroidSchedulers;

import static com.trello.rxlifecycle.android.ActivityEvent.PAUSE;

public class ServiceDiscoveryExampleActivity extends RxAppCompatActivity {

    @BindView(R.id.connect)
    Button connectButton;
    @BindView(R.id.scan_results)
    RecyclerView recyclerView;
    private DiscoveryResultsAdapter adapter;
    private String macAddress;
    private RxBleDevice bleDevice;

    public static final String EXTRA_MAC_ADDRESS = "extra_mac_address";
    public static final String EXTRA_CHARACTERISTIC_UUID = "extra_uuid";

    @OnClick(R.id.connect)
    public void onConnectToggleClick() {
        bleDevice.establishConnection(false)
            .flatMap(RxBleConnection::discoverServices)
            .first() // Disconnect automatically after discovery
            .compose(bindUntilEvent(PAUSE))
            .observeOn(AndroidSchedulers.mainThread())
            .doOnUnsubscribe(this::updateUI)
            .subscribe(adapter::swapScanResult, this::onConnectionFailure);

        updateUI();
    }
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_example3);
        ButterKnife.bind(this);
        macAddress = getIntent().getStringExtra(EXTRA_MAC_ADDRESS);
        ButterKnife.bind(this);
        //noinspection ConstantConditions
        getSupportActionBar().setSubtitle(getString(R.string.mac_address,
macAddress));
        bleDevice = MainActivity.rxBleClient.getBleDevice(macAddress);
        configureResultList();

        onConnectToggleClick();
    }

    private void configureResultList() {
        recyclerView.setHasFixedSize(true);
        LinearLayoutManager recyclerViewLayoutManager = new LinearLayoutManager(this);
        recyclerView.setLayoutManager(recyclerViewLayoutManager);
    }
}
```

```
        adapter = new DiscoveryResultsAdapter();
        recyclerView.setAdapter(adapter);
        adapter.setOnItemClickListener(view -> {
            final int childAdapterPosition =
recyclerView.getChildAdapterPosition(view);
            final DiscoveryResultsAdapter.AdapterItem itemAtPosition =
adapter.getItem(childAdapterPosition);
            onItemClick(itemAtPosition);
        });
    }
    private void onItemClick(DiscoveryResultsAdapter.AdapterItem item) {

        if (item.type == DiscoveryResultsAdapter.AdapterItem.CHARACTERISTIC) {
            MainActivity.characteristicUuid = item.uuid;
            MainActivity.macAddress = macAddress;
            final Intent intent = new Intent(this, MainActivity.class);
            startActivity(intent
                .addFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP |
Intent.FLAG_ACTIVITY_SINGLE_TOP));
            finish();
            startActivity(intent);
        } else {
            //noinspection ConstantConditions
            Snackbar.make(findViewById(android.R.id.content),
R.string.not_clickable, Snackbar.LENGTH_SHORT).show();
        }
    }
    private boolean isConnected() {
        if (bleDevice != null) {
            return bleDevice.getConnectionState() ==
RxBleConnection.RxBleConnectionState.CONNECTED;
        } else {
            return false;
        }
    }
    private void onConnectionFailure(Throwable throwable) {
        //noinspection ConstantConditions
        Snackbar.make(findViewById(android.R.id.content), "Connection error: " +
throwable, Snackbar.LENGTH_SHORT).show();
    }

    private void updateUI() {
        connectButton.setEnabled(!isConnected());
    }
}
```

ANNEX H

```
package com.polidea.rxandroidble.sample;

import android.content.Intent;
import android.os.Bundle;
import android.support.v7.app.AppCompatActivity;

import butterknife.ButterKnife;
import butterknife.OnClick;
```

```
public class DeviceActivity extends AppCompatActivity {

    public static final String EXTRA_MAC_ADDRESS = "extra_mac_address";
    private String macAddress;

    @OnClick(R.id.connect)
    public void onConnectClick() {
        final Intent intent = new Intent(this,
ServiceDiscoveryExampleActivity.class);
        intent.putExtra(EXTRA_MAC_ADDRESS, macAddress);
        startActivity(intent);
    }

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_device);
        ButterKnife.bind(this);
        macAddress = getIntent().getStringExtra(EXTRA_MAC_ADDRESS);
        //noinspection ConstantConditions
        getSupportActionBar().setSubtitle(getString(R.string.mac_address,
macAddress));
    }
}
```

ANNEX I

```
package com.polidea.rxandroidble.sample;

import android.bluetooth.BluetoothGattCharacteristic;
import android.bluetooth.BluetoothGattService;
import android.support.v7.widget.RecyclerView;
import android.text.TextUtils;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.TextView;
```

```
import com.polidea.rxandroidble.RxBleDeviceServices;

import java.util.ArrayList;
import java.util.List;
import java.util.UUID;

import butterknife.BindView;
import butterknife.ButterKnife;

class DiscoveryResultsAdapter extends
RecyclerView.Adapter<DiscoveryResultsAdapter.ViewHolder> {

    static class AdapterItem {

        static final int SERVICE = 1;
        static final int CHARACTERISTIC = 2;
        final int type;
        final String description;
        final UUID uuid;

        AdapterItem(int type, String description, UUID uuid) {
            this.type = type;
            this.description = description;
            this.uuid = uuid;
        }
    }

    static class ViewHolder extends RecyclerView.ViewHolder {

        @BindView(android.R.id.text1)
        TextView line1;
        @BindView(android.R.id.text2)
        TextView line2;

        ViewHolder(View itemView) {
            super(itemView);
            ButterKnife.bind(this, itemView);
        }
    }

    interface OnAdapterItemClickListener {

        void onAdapterViewClick(View view);
    }

    private final List<AdapterItem> data = new ArrayList<>();
    private OnAdapterItemClickListener onAdapterItemClickListener;
    private final View.OnClickListener onClickListener = new
View.OnClickListener() {
        @Override
        public void onClick(View v) {

            if (onAdapterItemClickListener != null) {
                onAdapterItemClickListener.onAdapterViewClick(v);
            }
        }
    }
}
```

```
};

@Override
public int getItemCount() {
    return data.size();
}

@Override
public int getItemViewType(int position) {
    return getItem(position).type;
}

@Override
public void onBindViewHolder(ViewHolder holder, int position) {
    final int itemViewType = holder.getItemViewType();
    final AdapterItem item = getItem(position);

    if (itemViewType == AdapterItem.SERVICE) {
        holder.line1.setText(String.format("Service: %s", item.description));
    } else {
        holder.line1.setText(String.format("Characteristic: %s",
item.description));
    }

    holder.line2.setText(item.uuid.toString());
}

@Override
public ViewHolder onCreateViewHolder(ViewGroup parent, int viewType) {
    final int layout = viewType == AdapterItem.SERVICE ?
R.layout.item_discovery_service : R.layout.item_discovery_characteristic;
    final View itemView =
LayoutInflater.from(parent.getContext()).inflate(layout, parent, false);
    itemView.setOnClickListener(onClickListener);
    return new ViewHolder(itemView);
}

void setOnAdapterItemClickListner(OnAdapterItemClickListner
onAdapterItemClickListner) {
    this.onAdapterItemClickListner = onAdapterItemClickListner;
}

void swapScanResult(RxBleDeviceServices services) {
    data.clear();

    for (BluetoothGattService service : services.getBluetoothGattServices()) {
        // Add service
        data.add(new AdapterItem(AdapterItem.SERVICE, getServiceType(service),
service.getUuid()));
        final List<BluetoothGattCharacteristic> characteristics =
service.getCharacteristics();

        for (BluetoothGattCharacteristic characteristic : characteristics) {
            data.add(new AdapterItem(AdapterItem.CHARACTERISTIC,
describeProperties(characteristic), characteristic.getUuid()));
        }
    }
}
```

```
        notifyDataSetChanged();
    }

    private String describeProperties(BluetoothGattCharacteristic characteristic)
    {
        List<String> properties = new ArrayList<>();
        if (isCharacteristicReadable(characteristic)) properties.add("Read");
        if (isCharacteristicWriteable(characteristic)) properties.add("Write");
        if (isCharacteristicNotifiable(characteristic)) properties.add("Notify");
        return TextUtils.join(" ", properties);
    }

    AdapterItem getItem(int position) {
        return data.get(position);
    }

    private String getServiceType(BluetoothGattService service) {
        return service.getType() == BluetoothGattService.SERVICE_TYPE_PRIMARY ?
"primary" : "secondary";
    }

    private boolean isCharacteristicNotifiable(BluetoothGattCharacteristic
characteristic) {
        return (characteristic.getProperties() &
BluetoothGattCharacteristic.PROPERTY_NOTIFY) != 0;
    }

    private boolean isCharacteristicReadable(BluetoothGattCharacteristic
characteristic) {
        return ((characteristic.getProperties() &
BluetoothGattCharacteristic.PROPERTY_READ) != 0);
    }

    private boolean isCharacteristicWriteable(BluetoothGattCharacteristic
characteristic) {
        return (characteristic.getProperties() &
(BluetoothGattCharacteristic.PROPERTY_WRITE
| BluetoothGattCharacteristic.PROPERTY_WRITE_NO_RESPONSE)) != 0;
    }
}
```

ANNEX J

```
package com.polidea.rxandroidble.sample;

import android.content.Intent;
import android.content.pm.ActivityInfo;
import android.os.Bundle;
import android.support.v7.widget.SwitchCompat;
import android.util.Log;
import android.view.View;
import android.widget.AdapterView;
import android.widget.Button;
import android.widget.EditText;
import android.widget.ListView;
import android.widget.TextView;
import android.widget.Toast;

import com.polidea.rxandroidble.RxBleClient;
```



```
import com.polidea.rxandroidble.RxBleConnection;
import com.polidea.rxandroidble.RxBleDevice;
import com.polidea.rxandroidble.sample.util.HexString;
import com.polidea.rxandroidble.utils.ConnectionSharingAdapter;
import com.trello.rxlifecycle.components.support.RxAppCompatActivity;

import java.io.FileOutputStream;
import java.util.ArrayList;
import java.util.List;
import java.util.UUID;

import butterknife.BindView;
import butterknife.ButterKnife;
import butterknife.OnClick;
import rx.Observable;
import rx.Subscription;
import rx.android.schedulers.AndroidSchedulers;
import rx.subjects.PublishSubject;

import static com.polidea.rxandroidble.sample.util.HexString.bytesToString;

public class MainActivity extends RxAppCompatActivity {
    Button bt1;
    Button bt3;
    Button bt4;
    EditText etdiam;
    TextView tvdiam;
    TextView lecturaBT;
    TextView con_state;

    ListView notifyListView;
    TextView connectionStateView;
    SwitchCompat autoConnectToggleSwitch;

    private static int diam;

    final static double k = 177;
    double force;
    double red;
    String filename = "salida.txt";
    FileOutputStream outputStream;

    public static RxBleClient rxBleClient;
    protected static RxBleDevice bleDevice;
    protected Subscription connectionSubscription;

    protected static List<String> changelist = new ArrayList<String>();

    public static String macAddress = null;
    public static UUID characteristicUuid;

    private PublishSubject<Void> disconnectTriggerSubject =
PublishSubject.create();
    private Observable<RxBleConnection> connectionObservable;
```

```
@BindView(R.id.connectBT)
Button connectBT;
@BindView(R.id.readOutput)
TextView readOutput;
@BindView(R.id.readBT)
Button readBT;
@BindView(R.id.writeBT)
Button writeBT;
@BindView(R.id.notifyBT)
Button notifyBT;

public int getDiam() {
    return diam;
}
public void setDiam(int diam) {
    this.diam = diam;
}

@Override
protected void onCreate(Bundle savedInstanceState) {
    setRequestedOrientation(ActivityInfo.SCREEN_ORIENTATION_PORTRAIT);
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    ButterKnife.bind(this);
    rxBleClient = SampleApplication.getRxBleClient(this);
    bt1 = (Button) findViewById(R.id.forceButton);
    bt3 = (Button) findViewById(R.id.saveButton);
    bt4 = (Button) findViewById(R.id.scanButton);
    etdiam = (EditText) findViewById(R.id.DIAM_BOX);
    tvdiam = (TextView) findViewById(R.id.textView2);
    lecturaBT = (TextView) findViewById(R.id.Lectura);
    notifyListView = (ListView) findViewById(R.id.notify_list);
    con_state = (TextView) findViewById(R.id.connection_state);
    ArrayAdapter<String> arrayAdapter = new ArrayAdapter<String>(
        this,
        android.R.layout.simple_list_item_1,
        changelist);
    notifyListView.setAdapter(arrayAdapter);
    bt1.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View view) {
            Intent bt1 = new Intent(MainActivity.this, plotforce.class);
            startActivity(bt1);
        }
    });
    bt3.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View view) {
            diam = Integer.valueOf(etdiam.getText().toString());
            String s = String.valueOf(diam);
            tvdiam.setText(s);
            Toast.makeText(MainActivity.this, "Diameter saved : " + getDiam(),
Toast.LENGTH_SHORT).show();
            etdiam.setText("");
        }
    });
}
```

```
});
bt4.setOnClickListener(new View.OnClickListener(){
    @Override
    public void onClick(View view){
        Intent bt4 = new Intent(MainActivity.this, ScanActivity.class);
        startActivity(bt4);
    }
});
etdiam.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View view) {
        etdiam.setText("");
    }
});
}
@OnClick(R.id.connectBT)
public void ConnectBT() {
    if (macAddress!= null & characteristicUuid!=null) {
        if (isConnected()) {
            triggerDisconnect();
            updateUIscan();
        } else {
            connectionObservable.subscribe(rxBleConnection -> {
                Log.d(getClass().getSimpleName(), "Hey, connection has been
established!");
                runOnUiThread(this::updateUIscan);
            }, this::onConnectionFailure);
        }
    }
}
@OnClick(R.id.readBT)
public void ReadBT() {
    if (isConnected()) {
        connectionObservable
            .flatMap(rxBleConnection ->
rxBleConnection.readCharacteristic(characteristicUuid))
            .observeOn(AndroidSchedulers.mainThread())
            .subscribe(bytes -> {
                readOutput.setText(bytesToString(bytes));
            }, this::onReadFailure);
    }
}
@OnClick(R.id.writeBT)
public void onWriteClick() {
    if (isConnected()) {
        connectionObservable
            .flatMap(rxBleConnection ->
rxBleConnection.writeCharacteristic(characteristicUuid, getInputBytes()))
            .observeOn(AndroidSchedulers.mainThread())
            .subscribe(bytes -> {
                onWriteSuccess();
            }, this::onWriteFailure);
    }
}
private byte[] getInputBytes() {
    return HexString.stringToBytes(readOutput.getText().toString());
}
```

```

        private boolean isConnected() {
            if (bleDevice != null) {
                return bleDevice.getConnectionState() ==
RxBleConnection.RxBleConnectionState.CONNECTED;
            } else {
                return false;
            }
        }
        @OnClick(R.id.notifyBT)
        public void NotifyBT() {
            if (isConnected()) {
                connectionObservable
                    .flatMap(rxBleConnection ->
RxBleConnection.setupNotification(characteristicUuid))
                    .doOnNext(notificationObservable ->
runOnUiThread(this::notificationHasBeenSetUp))
                    .flatMap(notificationObservable -> notificationObservable)
                    .observeOn(AndroidSchedulers.mainThread())
                    .subscribe(this::onNotificationReceived,
this::onNotificationSetupFailure);
            }
        }
        private void onNotificationReceived(byte[] bytes) {
            //noinspection ConstantConditions
            changelist.add(bytesToString(bytes));
            red = Double.valueOf(bytesToString(bytes));
            force = (red/k)*(diam/80)*1517;
            lecturaBT.setText(String.valueOf((force)));
        }

        @Override
        protected void onRestart() {
            super.onRestart();
            if (macAddress!=null && characteristicUuid!=null) {

                bleDevice = rxBleClient.getBleDevice(macAddress);
                connectionObservable = prepareConnectionObservable();
                //noinspection ConstantConditions
                getSupportActionBar().setSubtitle(getString(R.string.mac_address,
macAddress));
                ConnectBT();
                updateUIscan();

            }
        }
        private Observable<RxBleConnection> prepareConnectionObservable() {
            return bleDevice
                .establishConnection(false)
                .takeUntil(disconnectTriggerSubject)
                .doOnUnsubscribe(this::clearSubscription)
                .compose(new ConnectionSharingAdapter());
        }
        private void onConnectionFailure(Throwable throwable) {
            //noinspection ConstantConditions
        }
        private void onReadFailure(Throwable throwable) {

```

```

        //noinspection ConstantConditions
    }
    private void onWriteSuccess() {
        //noinspection ConstantConditions
    }
    private void onWriteFailure(Throwable throwable) {
        //noinspection ConstantConditions
    }
    private void onNotificationSetupFailure(Throwable throwable) {
        //noinspection ConstantConditions
    }
    private void notificationHasBeenSetUp() {
        //noinspection ConstantConditions
    }
    private void clearSubscription() {
        updateUIscan();
    }
    private void triggerDisconnect() {
        disconnectTriggerSubject.onNext(null);
    }
    protected void updateUIscan() {
        connectBT.setText(isConnected() ? getString(R.string.disconnect) :
"Connect");
        con_state.setText(isConnected() ? "Connected" : "Disconnected");
        readBT.setEnabled(isConnected());
        writeBT.setEnabled(isConnected());
        notifyBT.setEnabled(isConnected());
    }
}

```

ANNEX K

```

package com.polidea.rxandroidble.sample;

import android.os.Bundle;
import android.widget.TextView;

import com.jjoe64.graphview.GraphView;
import com.jjoe64.graphview.series.DataPoint;
import com.jjoe64.graphview.series.PointsGraphSeries;

public class plotforce extends MainActivity {

    final static double k = 177;
    TextView fMAX;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_plotforce);
        GraphView graph = (GraphView) findViewById(R.id.graphf);
        fMAX = (TextView) findViewById(R.id.textMAX);

        if (MainActivity.changelist.size() != 0) {
            PointsGraphSeries<DataPoint> series = new
PointsGraphSeries<>(generateData());

```

```

        graph.addSeries(series);
        series.setShape(PointsGraphSeries.Shape.POINT);
        series.setSize(10);//change the size of the point to a smaller one.
    }
}
private DataPoint[] generateData() {
    double max, force;
    int diam= getDiam();
    int count = MainActivity.changelist.size();
    DataPoint[] values = new DataPoint[count];
    max = Double.valueOf(MainActivity.changelist.get(0));
    for (int i=0; i<count; i++) {
        double x = i;
        double y = Double.valueOf(MainActivity.changelist.get(i));
        force = (y/k)*(diam/80)*1517;
        if (force > max) { max = force; }
        DataPoint v = new DataPoint(x, force);
        values[i] = v;
    }

    String maxString = "Fmax = " + Double.toString(max);
    fMAX.setText(maxString);

    return values;
}
}

```

ANNEX L

```

package com.polidea.rxandroidble.sample;

import android.app.Application;
import android.content.Context;

import com.polidea.rxandroidble.RxBleClient;
import com.polidea.rxandroidble.internal.RxBleLog;

public class SampleApplication extends Application {

    private RxBleClient rxBleClient;

    /**
     * In practise you will use some kind of dependency injection pattern.
     */
    public static RxBleClient getRxBleClient(Context context) {
        SampleApplication application = (SampleApplication)
context.getApplicationContext();
        return application.rxBleClient;
    }

    @Override
    public void onCreate() {
        super.onCreate();
        rxBleClient = RxBleClient.create(this);
        RxBleClient.setLogLevel(RxBleLog.DEBUG);
    }
}

```

```
}
}
```

ANNEX M

```
<?xml version="1.0" encoding="utf-8"?>
<manifest
    package="com.polidea.rxandroidble.sample"
    xmlns:android="http://schemas.android.com/apk/res/android">

    <application
        android:name=".SampleApplication"
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme">
        <activity
            android:name=".MainActivity"
            android:windowSoftInputMode="stateHidden"
            android:label="@string/app_name">
            <intent-filter>
                <action android:name="android.intent.action.MAIN"/>

                <category android:name="android.intent.category.LAUNCHER"/>
            </intent-filter>
        </activity>
        <activity android:name=".plotforce" />
        <activity
            android:name=".ScanActivity"
            android:label="@string/title_example2"/>
        <activity
            android:name=".ServiceDiscoveryExampleActivity"
            android:label="@string/title_example2"/>
        <activity android:name=".DeviceActivity"/>
    </application>
</manifest>
```

ANNEX N

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin">

    <Button
        android:id="@+id/scan_toggle_btn"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="@string/start_scan" />
```

```
<android.support.v7.widget.RecyclerView
    android:id="@+id/scan_results"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:scrollbars="vertical"/>
</LinearLayout>
```

ANNEX O

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin">

    <Button
        android:id="@+id/connect"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Connect & discover services"/>

    <android.support.v7.widget.RecyclerView
        android:id="@+id/scan_results"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:scrollbars="vertical"/>
</LinearLayout>
```

ANNEX P

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.constraint.ConstraintLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/activity_main"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <Button
        android:id="@+id/writeBT"
        android:layout_width="88dp"
        android:layout_height="wrap_content"
        android:enabled="false"
        android:text="Write"
        android:layout_marginEnd="16dp"
        android:layout_marginRight="16dp"
        app:layout_constraintRight_toRightOf="parent">
```



```
app:layout_constraintTop_toTopOf="@+id/readOutput"
android:layout_marginTop="0dp" />

<TextView
    android:id="@+id/connection_state"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    tools:text="@string/connection_state"
    android:layout_marginStart="24dp"
    android:layout_marginLeft="24dp"
    app:layout_constraintLeft_toLeftOf="@+id/connectStatus"
    android:layout_marginBottom="16dp"
    app:layout_constraintBottom_toTopOf="@+id/readOutput" />

<TextView
    android:id="@+id/FORCE_BOX"
    android:layout_width="99dp"
    android:layout_height="22dp"
    android:layout_marginStart="16dp"
    android:text="Force Sensor"
    android:textAppearance="@style/TextAppearance.AppCompat.Body2"
    android:textSize="16sp"
    android:textStyle="normal|bold|italic"
    tools:layout_editor_absoluteY="16dp"
    tools:layout_editor_absoluteX="16dp" />

<TextView
    android:id="@+id/Magnitud_F"
    android:layout_width="wrap_content"
    android:layout_height="34dp"
    android:layout_marginBottom="0dp"
    android:layout_marginLeft="8dp"
    android:layout_marginStart="16dp"
    android:layout_marginTop="8dp"
    android:maxLength="50dp"
    android:minHeight="154dp"
    android:scrollbars="vertical"
    android:text="N"
    android:textAlignment="viewStart"
    app:layout_constraintBottom_toBottomOf="@+id/Lectura"
    app:layout_constraintLeft_toRightOf="@+id/Lectura"
    app:layout_constraintTop_toBottomOf="@+id/FORCE_BOX"
    app:layout_constraintVertical_bias="0.588"
    tools:minWidth="50dp" />

<Button
    android:id="@+id/forceButton"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="PLOT"
    android:layout_marginRight="16dp"
    app:layout_constraintRight_toRightOf="parent"
    android:layout_marginEnd="16dp"
    app:layout_constraintBottom_toBottomOf="@+id/Magnitud_F" />
```

```
<Button
    android:id="@+id/scanButton"
    android:layout_width="88dp"
    android:layout_height="48dp"
    android:layout_marginEnd="16dp"
    android:layout_marginRight="16dp"
    android:text="SCAN"
    app:layout_constraintRight_toRightOf="parent"
    tools:layout_conversion_absoluteHeight="48dp"
    tools:layout_conversion_absoluteWidth="88dp"
    tools:layout_conversion_absoluteX="275dp"
    tools:layout_conversion_absoluteY="209dp"
    app:layout_constraintTop_toTopOf="@+id/connectBT" />

<Button
    android:id="@+id/saveButton"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginEnd="16dp"
    android:text="Save"
    tools:layout_constraintRight_creator="1"
    app:layout_constraintTop_toTopOf="@+id/DIAM_BOX"
    android:layout_marginRight="16dp"
    app:layout_constraintRight_toRightOf="parent" />

<EditText
    android:id="@+id/DIAM_BOX"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginLeft="16dp"
    android:layout_marginStart="16dp"
    android:ems="10"
    android:hint="Enter diameter"
    android:inputType="number"
    app:layout_constraintLeft_toLeftOf="parent"
    tools:focusable="false"
    tools:layout_constraintLeft_creator="1"
    android:layout_marginBottom="8dp"
    app:layout_constraintBottom_toTopOf="@+id/textView2" />

<TextView
    android:id="@+id/textView2"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginLeft="16dp"
    android:layout_marginStart="16dp"
    android:text="DIAMETRO"
    app:layout_constraintLeft_toLeftOf="parent"
    tools:layout_constraintLeft_creator="1"
    android:layout_marginBottom="16dp"
    app:layout_constraintBottom_toTopOf="@+id/connectBT" />

<TextView
    android:id="@+id/connectStatus"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
```

```
android:layout_marginBottom="8dp"
android:layout_marginEnd="8dp"
android:layout_marginLeft="8dp"
android:layout_marginRight="8dp"
android:layout_marginStart="16dp"
android:text="Connection?"
android:textAppearance="@style/TextAppearance.AppCompat.Body2"
android:textSize="18sp"
android:textStyle="bold"
app:layout_constraintBottom_toTopOf="@+id/connection_state"
app:layout_constraintHorizontal_bias="0.0"
app:layout_constraintLeft_toLeftOf="parent"
app:layout_constraintRight_toLeftOf="@+id/notifyBT"
tools:layout_constraintLeft_creator="1" />
```

<Button

```
android:id="@+id/connectBT"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="Connect BT"
android:layout_marginLeft="16dp"
app:layout_constraintLeft_toLeftOf="parent"
android:layout_marginBottom="16dp"
app:layout_constraintBottom_toTopOf="@+id/connectStatus"
android:layout_marginStart="16dp" />
```

<Button

```
android:id="@+id/readBT"
android:layout_width="88dp"
android:layout_height="wrap_content"
android:layout_marginStart="16dp"
android:enabled="false"
android:text="Read"
android:visibility="invisible"
app:layout_constraintLeft_toRightOf="@+id/connectBT"
android:layout_marginLeft="-13dp"
app:layout_constraintTop_toTopOf="@+id/connectBT"
android:layout_marginTop="0dp" />
```

<EditText

```
android:id="@+id/readOutput"
android:layout_width="0dp"
android:layout_height="41dp"
android:layout_marginBottom="21dp"
android:layout_marginEnd="16dp"
android:layout_marginLeft="16dp"
android:layout_marginRight="16dp"
android:layout_marginStart="16dp"
android:hint="Write 1 to begin"
android:textSize="14sp"
app:layout_constraintBottom_toBottomOf="parent"
app:layout_constraintHorizontal_bias="1.0"
app:layout_constraintLeft_toLeftOf="parent"
app:layout_constraintRight_toLeftOf="@+id/writeBT" />
```

<Button

```
android:id="@+id/notifyBT"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:enabled="false"
android:text="Register notifications"
android:layout_marginRight="8dp"
app:layout_constraintRight_toRightOf="parent"
app:layout_constraintTop_toTopOf="@+id/connectStatus"
android:layout_marginEnd="16dp"
android:layout_marginTop="0dp" />
```

```
<TextView
```

```
    android:id="@+id/Lectura"
    android:layout_width="wrap_content"
    android:layout_height="34dp"
    android:layout_marginLeft="0dp"
    android:layout_marginTop="8dp"
    android:text="Lectura"
    app:layout_constraintLeft_toLeftOf="@+id/FORCE_BOX"
    app:layout_constraintTop_toBottomOf="@+id/FORCE_BOX"
    tools:layout_editor_absoluteX="17dp"
    tools:layout_editor_absoluteY="41dp" />
```

```
<ListView
```

```
    android:id="@+id/notify_list"
    android:layout_width="0dp"
    android:layout_height="0dp"
    android:layout_marginBottom="16dp"
    android:layout_marginEnd="16dp"
    android:layout_marginLeft="16dp"
    android:layout_marginRight="16dp"
    android:layout_marginStart="16dp"
    android:layout_marginTop="8dp"
    android:scrollbarAlwaysDrawHorizontalTrack="false"
    android:scrollbarAlwaysDrawVerticalTrack="false"
    android:scrollbars="none|vertical"
    app:layout_constraintBottom_toTopOf="@+id/DIAM_BOX"
    app:layout_constraintHorizontal_bias="0.0"
    app:layout_constraintLeft_toLeftOf="parent"
    app:layout_constraintRight_toRightOf="parent"
    app:layout_constraintTop_toBottomOf="@+id/Lectura"
    app:layout_constraintVertical_bias="1.0" />
```

```
<TextView
```

```
    android:id="@+id/MAGN_UR"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="UR*10000"
    app:layout_constraintRight_toLeftOf="@+id/forceButton"
    android:layout_marginRight="8dp"
    app:layout_constraintTop_toTopOf="parent"
    android:layout_marginTop="60dp" />
```

```
</android.support.constraint.ConstraintLayout>
```

ANNEX Q

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.constraint.ConstraintLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/activity_plotforce"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context="com.polidea.rxandroidble.sample.plotforce">

    <com.jjoe64.graphview.GraphView
        android:id="@+id/graphf"
        android:layout_width="0dp"
        android:layout_height="0dp"
        android:layout_marginBottom="8dp"
        app:layout_constraintBottom_toTopOf="@+id/textMAX"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintTop_toTopOf="parent"
        tools:layout_constraintTop_creator="1" />
    <TextView
        android:id="@+id/textMAX"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:layout_marginEnd="8dp"
        android:layout_marginLeft="8dp"
        android:layout_marginRight="8dp"
        android:layout_marginStart="8dp"
        android:contentDescription="Función Máximo"
        android:singleLine="false"
        android:textAlignment="center"
        android:textSize="18sp"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintHorizontal_bias="0.01"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        tools:layout_constraintLeft_creator="1"
        android:layout_marginBottom="-1dp" />
</android.support.constraint.ConstraintLayout>
```

ANNEX R

```
/*
The circuit:
* RX is digital pin 10 (connect to TX of other device)
* TX is digital pin 11 (connect to RX of other device)
Note:
Not all pins on the Mega and Mega 2560 support change interrupts,
so only the following can be used for RX:
10, 11, 12, 13, 50, 51, 52, 53, 62, 63, 64, 65, 66, 67, 68, 69
Not all pins on the Leonardo support change interrupts,
so only the following can be used for RX:
8, 9, 10, 11, 14 (MISO), 15 (SCK), 16 (MOSI).
```

```
*/
#include <SoftwareSerial.h>
int cont=0;
SoftwareSerial mySerial(10, 11); // RX, TX
Int
myArray[855]={94,93,93,93,93,92,92,93,93,93,93,93,93,93,93,93,94,96,98,101,102,
102,102,103,103,102,102,103,103,103,103,104,105,107,110,114,119,126,132,136,13
8,138,139,139,139,139,139,139,139,139,139,139,140,146,156,166,175,186,197,
209,224,242,253,257,259,262,263,263,264,265,266,267,267,267,268,268,268,268,267,26
9,274,281,291,301,313,324,337,351,367,385,403,422,440,459,475,492,511,526,534,539,
542,544,546,547,548,549,550,551,551,551,552,553,553,554,554,554,554,554,555,55
5,556,556,556,556,556,557,557,556,556,556,557,557,557,558,557,558,558,558,557,
557,558,558,558,558,558,558,558,558,558,558,558,560,562,566,570,573,577,58
2,589,598,606,614,622,630,640,653,671,690,708,728,747,765,779,794,810,825,840,855,
868,880,891,899,904,908,910,912,914,915,916,926,926,926,925,925,926,926,926,92
6,926,926,927,926,926,926,927,927,927,927,927,927,927,926,926,926,926,927,927,
927,927,927,927,927,926,926,927,927,926,926,926,926,927,927,926,926,927,927,92
6,927,927,927,927,927,927,927,927,927,927,927,927,927,927,926,926,926,927,926,
926,926,926,927,926,925,925,925,926,926,926,927,928,928,928,929,929,931,933,935,93
7,939,941,943,945,947,949,951,952,955,958,962,966,971,977,983,988,994,1002,1008,10
14,1019,1024,1028,1044,1051,1059,1067,1075,1084,1092,1100,1107,1112,1117,1122,1126
,1128,1131,1133,1134,1136,1137,1137,1139,1140,1140,1140,1141,1142,1142,1142,1143,1
144,1144,1144,1144,1144,1144,1145,1145,1145,1146,1146,1146,1146,1146,1146,114
6,1147,1147,1148,1148,1148,1149,1149,1149,1149,1150,1150,1150,1150,1149,1149,1151,
1150,1150,1150,1151,1151,1151,1151,1151,1151,1151,1156,1157,1158,1158,1158,1159,11
60,1163,1166,1169,1173,1176,1179,1181,1185,1188,1191,1193,1196,1198,1199,1201,1203
,1205,1207,1210,1213,1220,1227,1233,1238,1243,1247,1251,1253,1257,1262,1266,1270,1
274,1281,1289,1296,1302,1308,1314,1321,1328,1333,1337,1341,1344,1348,1353,1356,135
8,1361,1363,1365,1366,1367,1368,1369,1370,1370,1371,1371,1372,1383,1383,1382,1382,
1382,1382,1383,1408,1421,1432,1440,1445,1449,1450,1452,1453,1454,1455,1455,1455,14
55,1456,1457,1458,1459,1459,1459,1460,1461,1461,1462,1465,1465,1454,1427,1389,1365
,1358,1356,1356,1355,1354,1354,1354,1354,1353,1353,1346,1308,1238,1194,1183,1181,1
179,1177,1177,1176,1174,1173,1174,1173,1173,1174,1173,1173,1172,1172,1171,1170,117
0,1171,1171,1171,1171,1170,1171,1170,1170,1169,1169,1169,1169,1168,1167,1168,1169,
1168,1167,1156,1140,1132,1129,1127,1125,1124,1124,1123,1122,1122,1122,1121,1120,11
17,1088,1020,928,818,717,642,596,573,563,558,555,553,552,550,549,548,547,544,520,4
41,290,177,147,139,135,132,130,129,127,126,125,124,123,123,123,122,122,121,121,120
,120,119,118,117,118,118,118,117,117,117,117,117,116,115,115,122,122,121,122,121,1
21,121,122,122,122,121,124,129,132,136,141,148,155,162,168,174,179,184,190,198,207
,217,227,235,244,254,265,277,291,307,324,343,363,385,409,433,454,467,475,480,482,4
85,487,489,496,496,496,497,496,496,496,497,497,497,497,500,505,510,514,516,519,521
,524,526,527,529,530,531,532,532,532,533,534,535,536,535,536,537,537,537,545,546,5
46,546,546,545,545,547,550,552,554,557,560,565,573,583,595,610,626,641,656,671,686
,699,714,728,743,758,773,789,806,826,844,859,873,887,900,910,920,926,931,934,937,9
39,941,942,944,945,946,946,946,946,948,948,948,949,950,950,950,950,950,950,951,952
,953,953,953,953,953,953,954,956,959,963,967,971,976,982,992,1002,1012,1022,1035,1
049,1062,1073,1083,1094,1105,1118,1130,1142,1153,1162,1173,1186,1202,1217,1228,123
7,1243,1248,1251,1254,1256,1258,1259,1260,1261,1262,1262,1262,1263,1264,1265,1265,
1266,1266,1267,1267,1267,1267,1267,1267,1267,1268,1269,1269,1269,1269,1269,1269,12
69,1269,1270,1270,1270,1271,1271,1271,1271,1270,1270,1270,1271,1271,1271,1271,1272
,1272,1272,1271,1271,1271,1272,1272,1272,1272,1273,1273
};
void setup() {
  // Open serial communications and wait for port to open:
  Serial.begin(57600);
  // set the data rate for the SoftwareSerial port
  mySerial.begin(9600);
```

```
}

void loop() {
  // run over and over
  if (mySerial.available() && cont < 855) {
    int i;
    int k;
    for (i = 0 ; i < 855; i++){
      k = myArray[i];
      mySerial.print(k);
      cont++;
      Serial.print(cont);
      delay(100);
    }
  }
  if (cont > 855){
    software_Reset();
  }
}

void software_Reset() // Restarts program from beginning but does not reset the
peripherals and registers
{
  asm volatile (" jmp 0");
}
```