

Abstract

The objective of this project is to create a library of functions and procedures for Microsoft Excel. This library is used to treat data generated by Internet of Things devices.

To accomplish this goal, it's necessary to bridge the gap existing between the data generated and the spreadsheet program of choice that could easily handle this data, in this case Microsoft Excel.

This missing link comes in the form of a library written in "Visual Basic of Applications" (VBA), the language that Microsoft Excel uses for user defined operations.

This project contains the background and state of the art of the technologies involving the "Internet of Things" at the moment of its conception.

An Evaluation Board is used to emulate the behavior of different possible devices that could belong to the Internet of Things category.

The project contains explanations of the main concepts related to the "Internet of Things", it also explores and discusses how to best approach the conception of the library. Once the objectives are narrowed, it presents the code implemented with detailed explanation of the structures used.

The main objective is to properly retrieve and present data obtained with the evaluation board. This is accomplished by structuring a library with four modules related to each other. These modules are used to: (1) retrieve the data information using Http requests, (2) format the data in a way that is useful to treat and interpret, (3) create a program that automates these procedures and presents data reports that make it easy to manage data, and finally (4) present library description and provide help to the user so doubts and issues encountered can be solved.

After the library code is explained, validation tests are made to demonstrate the performance of the processes and functions contained in the library.

The result of the project is a solid library written in "Visual Basic for Applications", composed by four modules that can be upgraded or changed individually to perform new custom applications.

This library accomplishes the objective of providing a useful tool to both users and researchers.

Table of contents

ABSTRACT	1
TABLE OF CONTENTS	3
1. PREFACE	7
1.2. Project origin	7
1.3. Motivation	7
1.4. Prerequisites	8
2. INTRODUCTION	9
2.1. Project Objectives	9
2.2. Long term goal	9
3. STATE OF THE ART	11
3.1. The Internet of Things	11
3.2. Low Power Wide Area Network	14
3.3. SIGFOX technology	18
3.4. EVB TD 1204	20
3.5. IoT device emulation	22
4. LIBRARY DEVELOPMENT	26
4.1. Research: API, SENSOR, JSON language and VBA language.....	26
4.2. Desired functionality of the library.....	31
4.3. Code structure and implementation.....	40
4.3.1. General considerations.....	40
4.3.2. TDdeviceAPI library	42
4.3.3. Program structure	49
4.3.4. Report format.....	55
4.3.5. User interface	56
4.3.6. TDfunctionsDescriptions.....	59
4.4. Library diagram	60
5. TEST AND VALIDATION	62
5.1. TDdeviceAPI functions.....	62
5.2. Report Generation Program.....	63
5.3. User interface parameters	65
5.4. Miscellanea	66

6. BUDGET	67
7. ENVIRONMENTAL IMPACT	68
CONCLUSIONS	69
ACKNOWLEDGMENTS	71
BIBLIOGRAPHY	73

1. Preface

1.2. Project origin

The idea of this project was born after the realization that most data generated by devices connected to the internet of things networks lacked interpretation and study. In particular, the evaluation boards used in research projects, where the data study should be a major part to arrive at conclusions with confidence.

Therefore, there is a need to create or use an existing tool to be able to treat this data. This tool chosen is “Microsoft Excel”, which can be useful because of its widespread knowledge and easy access.

To reconcile the idea of using “Microsoft Excel” to treat the data created by the devices connected to IoT networks is necessary to bridge the gap between these two.

The missing element is a library of functions and procedures, written in the “Microsoft Excel” language: Visual Basic for Applications (VBA).

1.3. Motivation

The main motivation of this project is to gain insight of the raising technologies revolving the Internet of Things. This motivation comes from the realization that the tendency of the world is to interconnect everything, to either gain control over things or optimize procedures and requirements.

It is also worth noticing that this kind of new technologies are well accepted and assimilated by the majority of population, making it a good field to innovate and produce new ideas.

Even though some people might argue there is no need to dedicate efforts to develop this kind of technologies, I believe that researching every type of new technology aids in the ongoing process of acquiring knowledge, and at worst, makes it clear that certain line of work is not to be continued, only to discover a better path to be followed.

The possibilities are infinite and most of the scenarios that could arise by developing this kind of technologies could take huge leaps in reduction of energy consumption, monitoring

health, reduce environmental and economic cost of logistics followed by a plethora of benefits and utilities.

1.4. Prerequisites

In order to conceive this project, it is necessary to possess an understanding of the state of the art of the technologies involved, such as “low power-wide area” networks (e.g. SIGFOX), emission of messages using this technology, gateways and device operational features and structure.

It was also necessary to achieve a high degree of understanding of the language the library is to be written in: Visual Basic for Applications, as well as a fluent use of Microsoft Excel and achieving even higher competence is tied to learning the language above mentioned.

Furthermore, it is required to have access to the support material used. The hardware used is an evaluation board (EVB TD 1204) that emulates the behavior of some devices that could be used in real applications related to the IoT.

Finally, it is also essential to understand how this evaluation board operates and how to use it correctly.

2. Introduction

2.1. Project Objectives

The objective of this project is to create an Excel VBA library for the IoT (Internet of Things). The successful creation of this library will be determined upon functionality and reliability of the code implemented. The code structure must be solid and support different operational conditions.

Additional objectives are the test and validation of all the structures contained in the designed in a library as well as providing explanation of the library within the code and outside the code.

As a consequence, the VBA library has the purpose of aiding future users of devices connected to Internet of Things networks. It should be easy to understand and use to both advanced users and beginners.

Another objective is to get to understand how the data is generated and retrieved by the devices connected to Internet of Things networks.

To achieve this objectives, comprehension of the “Evaluation Board TD1204” will be quintessential. This evaluation board contains a microcontroller and several external devices that generate data to simulate the behavior expected in an actual device.

The creation of applications in a network environment or the creation of Internet communications applications is excluded from the original objective list.

Thus focusing mainly on data retrieval and data presentation.

2.2. Long term goal

The final goal of this project is to provide a tool for users of devices that belong to the Internet of Things, as well as provide a framework to develop new projects related to the IoT.

If the project succeeds, users of mentioned devices should be able to comfortably treat data generated by their devices using the widely known worksheet program “Microsoft Excel”.

The project should be able to provide either a starting point, or a tool to be used in future projects involving the IoT.

3. State of the Art

3.1. The Internet of Things

The Internet of Things is a broad concept that is used to define the internetworking of physical devices, vehicles, buildings and other items. These items are embedded with electronics, software, sensors, actuators, and network connectivity that enable these objects to collect and exchange data.

"Things," in the IoT sense, can refer to a wide variety of devices such as heart monitoring implants, biochip transponders on farm animals, electric clams in coastal waters, automobiles with built-in sensors, DNA analysis devices for environmental/food/pathogen monitoring or field operation devices that assist firefighters in search and rescue operations.

These devices collect useful data with the help of various existing technologies and then autonomously flow the data between other devices.

This kind of data flow between devices is known as machine to machine communications (M2M).

The range of possibilities that the Internet of Things may offer is enormous and many fields can incorporate elements of it to increase efficiency and enhance performance in many problems that may arise.

Some of the current fields that are developing applications related to the Internet of Things are the following:

- 1- Smart Home: e.g. Smart Thermostat, Smart Fridge, Connected Lights, Smart Door lock.
- 2- Wearables: e.g. Smart Watch, Activity Tracker, Smart Glasses.
- 3- Smart City: e.g. Smart Parking and Smart Waste Management.
- 4- Smart Grid: e.g. Smart Metering.
- 5- Industrial Internet: e.g. Remote Asset Control.
- 6- Connected Car: e.g. Remote Car Control.

- 7- Connected Health: e.g. Cardiac Pacemaker monitoring.
- 8- Others fields like Smart Retail, Smart Supply-Chain and Smart Farming.

A quotidian example of what the Internet of Things may offer if fully operational is shown in the next figure (Figure 4-1.).

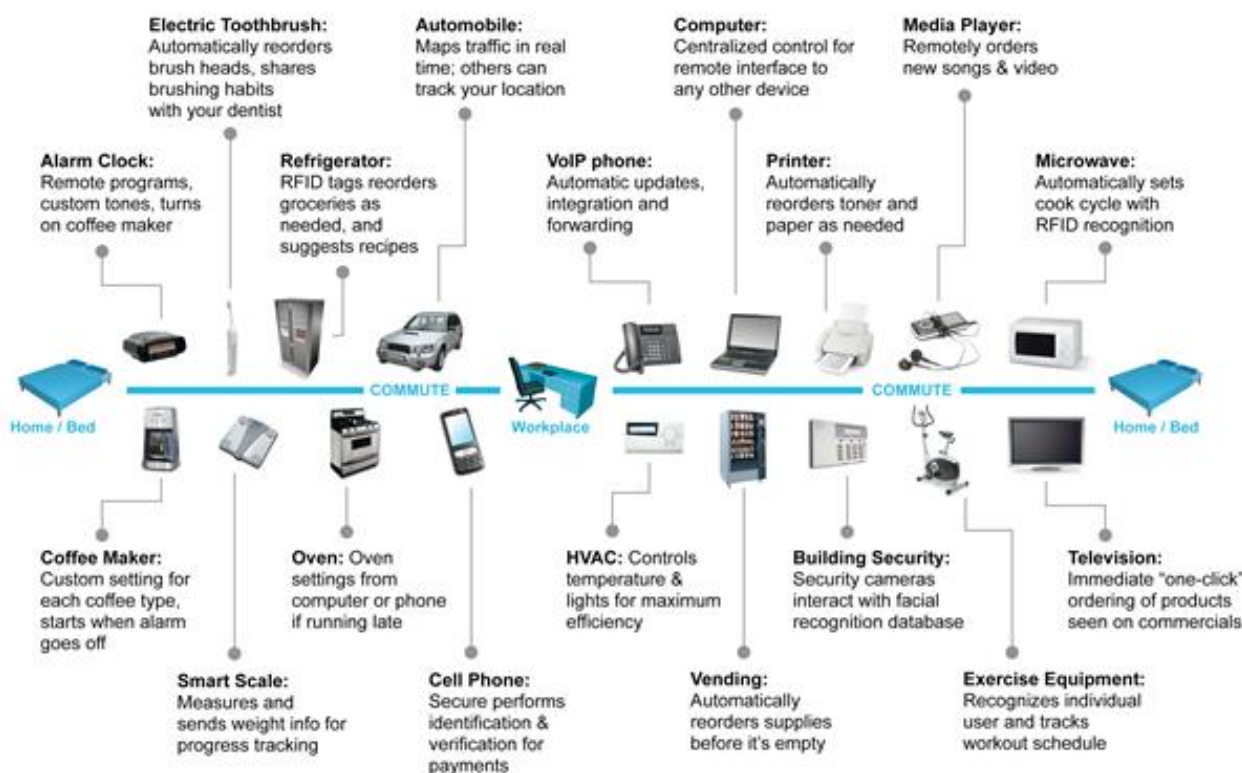


Figure 3-1. Internet of Things list. [1]

Figure 4-1. describes a day cycle where the devices collect and send data to optimize various procedures.

This procedures and operations could be automated and treated by the user by using a remote device, for example a smartphone.

Using a remote device, the user could control all kinds of devices. The following figure (Figure 4-2.) shows a model of communication between the user and the devices.

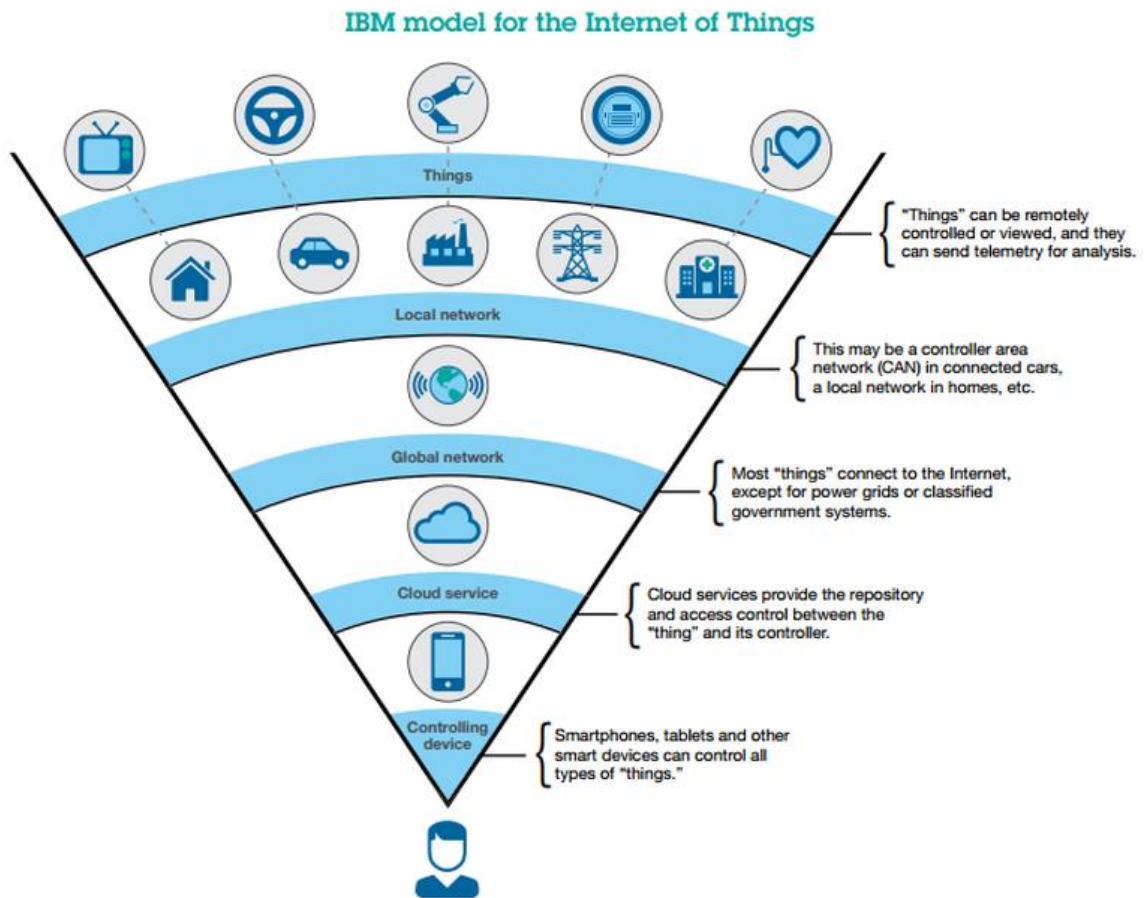


Figure 3-2. IBM model for the Internet of Things [2]

To sum up, the Internet of Things is a broad term that involves the M2M communications between devices and the applications developed in many fields with the purpose of using data to optimize processes and increase efficiency in any way possible.

3.2. Low Power Wide Area Network

M2M and IoT will give rise to billions of nodes that require connecting. Most of these will require only low bandwidth to transfer small amounts of data. Some will also require this to be connected over distances greater than those achievable simply by a transmitter on its own.

For many of these applications, the traditional cellular phone systems are too complex to allow low power operation, and too costly to be feasible for many small nodes (devices). In contrast, Low Power Wide Area Network technologies are a proper way to achieve low power operation of the devices.

The requirements of these kind of devices (IoT devices) must be supported by the network chosen.

One fundamental characteristic of these type of devices is very low power consumption which ensures a large lifetime without the need to change or recharge batteries. This is essential characteristic since one of the premises of the Internet of Things is the ability to control “Things” from afar and/or ensure that M2M communications are not interrupted.

Another important characteristic is the ability to have long range transmissions since some applications may involve collecting and sending data from devices that are far from one another. It is also important to have excellent geographic coverage, even in rural areas, for the same reasons that it is important to have long range transmissions.

Furthermore, because of the nature of the M2M communications in the IoT, the amount of information inside a message can be small (a few hundred bit/s or less).

Different wireless technologies cover different applications regarding range and bandwidth (Figure 4-3.). Long-range applications with low bandwidth requirements commonly found in IoT and M2M scenarios are not well supported by existing technologies (look Figure 4-3.). LPWAN technologies target these emerging applications and markets.

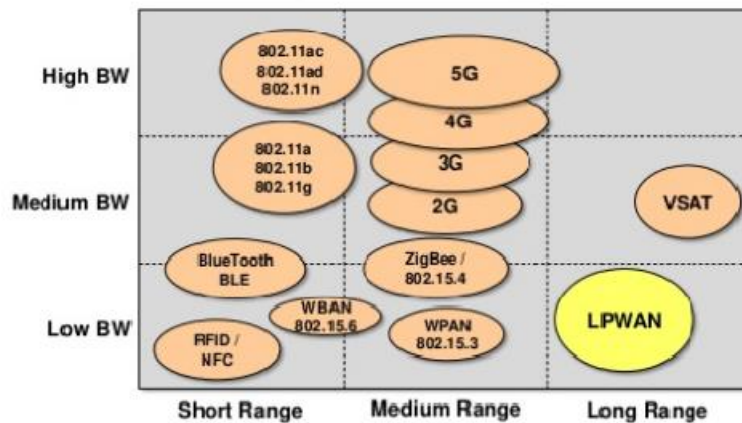


Figure 3-3. Range and Bandwidth array of networks. [3]

LPWAN technologies support the requirements needed for IoT communications. The advantages of direct connectivity or gateway connectivity as explained in “*Overview of Emerging Technologies for low power wide area networks in internet of things and M2M scenarios.*” [3] are:

The wireless portion of LPWAN networks uses a star topology. This obviates the need for complicated wireless mesh routing protocols which would greatly complicate the implementation of end devices and drive up power consumption.

- A. Direct device connectivity (base station):
 - 1- A base station provides connectivity to a large number of devices.
 - 2- The traffic is backhauled to servers (cloud) through TCP/IP based networks (Internet).
 - 3- The base station is responsible for protocol translation from IoT protocols such as MQTT or CoAP to device application protocols.
- B. Indirect device connectivity through a LPWAN gateway:
 - 1- In setups where devices cannot be directly reached through LPWAN, a local gateway bridges LPWAN connectivity to some short range radio (SRD) technology (e.g. ZigBee, BLE).

- 2- The gateway usually runs on mains power since it serves a larger number of devices and must convert between LPWAN and SRD radio technologies and protocols.
- 3- Gateways may help to improve security, since more powerful security algorithms can be implemented on the gateway than on the constrained devices.

In the still evolving IoT and M2M markets, a few competing radio technologies are emerging. These kind of technologies have in common the use of frequencies lower than 2.4GHz or 5.8GHz to achieve better penetration into buildings or underground installations.

Work is still in progress in most of these technologies but the ability to have long-ranged emissions, big geographical coverage and allow little power consumption, make this kind of technologies a good candidate for IoT communications.

Some of the LPWAN technologies are:

“*Greenwaves*”: a low-power, long range offering.

“*Haystack*”: a DASH7 low-power wireless network development platform.

“*LoRaWAN*”: LoRa Alliance’s Long Range WAN.

“*LTE-MTC*”: a development of LTE communications for connected things.

“*RPMA*”: On-Ramp Wireless’s Random Phase Multiple Access.

“*Symphony Link*”: from Link Lab.

“*ThingPark Wireless*”: Actility’s development of the LoRaWAN.

“*UNB (Ultra Narrow Band)*”: from various companies including Telensa, NWave and Sigfox.

“*Weightless*”: from the Weightless SIG.

“*WAVIoT*”: Narrowband M2M protocol.

Each one of them has different operational procedures, and some parameters like bandwidth and power consumption may vary slightly from one to another. Furthermore, some have open standards while others don’t, for example LoRaWAN uses “Semtech” chips only.

The LPWAN technology used in this project is SIGFOX technology, which uses UNB (Ultra narrow band). This is due to the fact that the evaluation board used to emulate the IoT devices send data to SENSOR, which is a platform of “Telecom Design”, that uses the SIGFOX network.

3.3. SIGFOX technology

SIGFOX technology's name is given after the French company that created it ("SIGFOX").

SIGFOX employs a cellular style system that enables remote devices to connect using ultra-narrow band (UNB) technology [4].

When a message is sent from the device, there is no signaling, nor negotiation between the device and the receiving station. The device decides when to send the message, picking up a pseudo-random frequency.

It's up to the network to detect the incoming messages, as well as validating & reduplicating them.

The message is then available in the SIGFOX cloud, and forwarded to any third party cloud platform chosen by the user. In this project, SENSOR platform from Telecom Design is chosen as the third party cloud.

The SIGFOX network is aimed at providing connectivity for a variety of applications and users.

The SIGFOX network performance is characterized by the following:

- 1- Up to 140 messages per object per day.
- 2- Payload size for each message is 12 bytes.
- 3- Wireless throughput up to 100 bits per second.

However, these limitations do not prevent the transmission of coded messages to implement actions at the backend, due the fact that with 12 bytes one can represent plenty of coded actions.

This fact provides either a way to exchange simple information, or complex codes that can be decoded at the backend.

The SIGFOX network is relying on Ultra-Narrow Band(UNB) modulation, and operating in unlicensed sub-GHz frequency bands [5].

This protocol offers a great resistance to jamming & standard interferers, as well as a great capacity per receiving base stations.

SIGFOX complies with local regulations, adjusting central frequency, power output and duty cycles.

The following table shows which frequency and regulation is applied in each region: (Figure 4-4.).

Region	Frequency	Regulation
Europe, Middle East - Radio Zone 1	868MHz	ETSI 300-220
North America - Radio Zone 2	902MHz	FCC part 15
South America, Australia, New Zealand - Radio Zone 4	920MHz	ANATEL 506, AS/NZS 4268

Figure 3-4. SIGFOX network frequency and regulation[5]

Regarding security of the information sent, every message is signed with information unique to the device (including a unique private key) and to the message itself.

Encryption and scrambling of the data are supported by the technology.

The information sent with the device will be accessed through the platform SENSOR of the third party chosen: TD (Telecom Design).

3.4. EVB TD 1204

To emulate the behavior of an IoT device, an evaluation board containing a SIGFOX gateway will be used. This evaluation board contains the gateway TD 1204, which was designed by Telecom Design, and manufactured and supplied by “AVNET Silica”.

TD1204 devices are high performance, low current SIGFOX gateways, RF transceiver and GPS receiver.

The TD1204 device versatility provides the gateway function from a local Narrow Band ISM network to the long-distance Ultra Narrow Band SIGFOX network.

The TD1204 device offers a RF sensitivity of -126 dBm while providing an output power of up to $+14$ dBm.

The TD1204 device versatility provides the gateway function from a local Narrow Band ISM network to the long-distance Ultra Narrow Band SIGFOX network.

Moreover, the fully integrated on-board GPS receiver combines sensitivity with ultralow Power.

The next figure shows a detail of the TD1204 on the evaluation board (Figure 4-5).

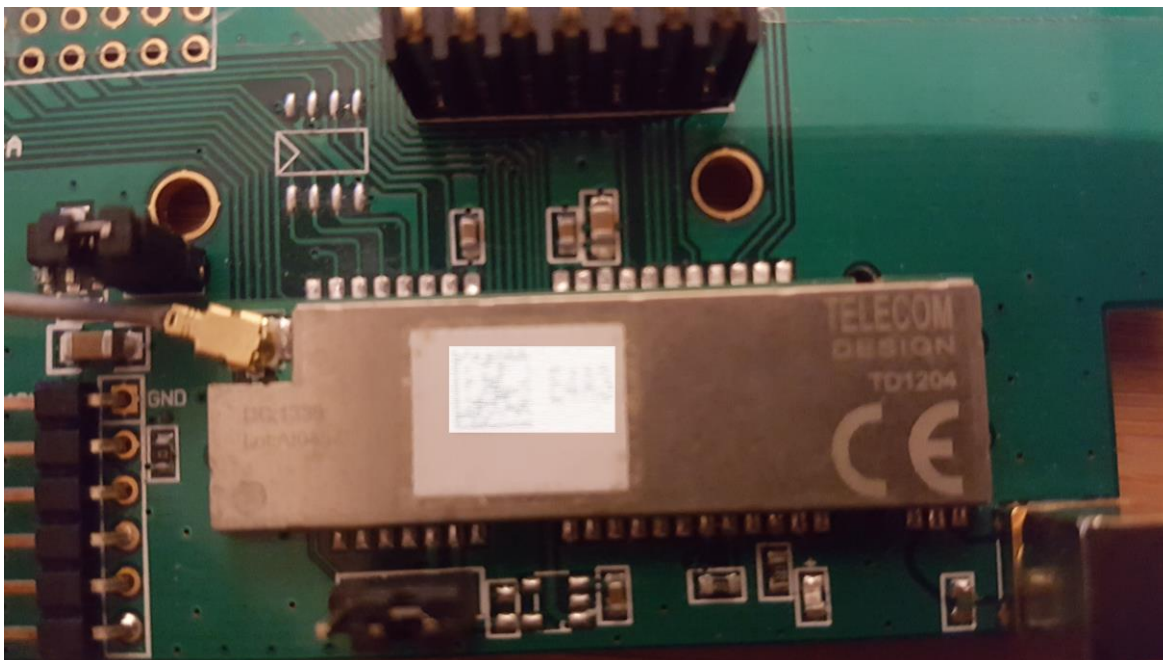


Figure 3-5. TD1204. [own source]

The Evaluation Board of the TD1204 incorporates:

- An ARM Cortex M3 processor of 32 bits banded base.
- A RF transceiver.
- High efficiency GPS receiver
- 3D accelerometer with movement detector and free fall detector.
- Digital and Analog interface.
- Low potency LVTTTL UART
- I2C Bus.
- Timers capable of counting pulses(input) or PWM(output).
- Two High Resolution A/D converter ad one D/A converter.
- Many input and output pins (GPIO).

The next figure shows the Evaluation Board EVB TD1204 (Figure 4-6.).

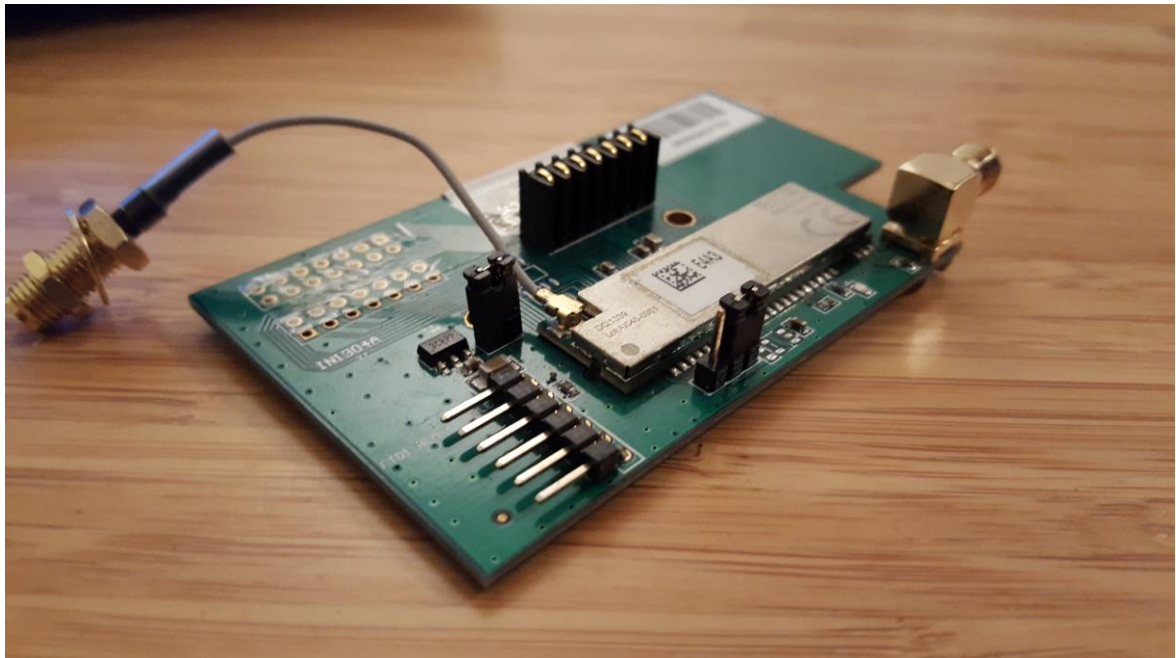


Figure 3-6. EVB TD1204. [own source]

3.5. IoT device emulation

The EVB TD 1204 is used to simulate the behavior of IoT devices.

To begin with, the EVB TD1204 must be connected to the terminal from where it will receive the orders. In this project, a PC was used as the terminal.

In order to send messages, I installed an FDTI VPC driver, which allowed me to use the USB port as an VPC (Virtual Port Channel).

I used the FDTI VPC driver that was suggested by Telecom Design [6].

The EVB TD1204 is connected using the USB port of the PC, (Figure 4-7).



Figure 3-7. EVB TD1204 connected to the PC with antenna. [own source]

Then I installed “PuTTY”, which is a HyperTerminal session program. Using “PuTTY” I was able to communicate with the EVB using the VPC created by the FDTI VPC driver. I chose the Serial option in the communications interface of “PuTTY” (Figure 4-8.).

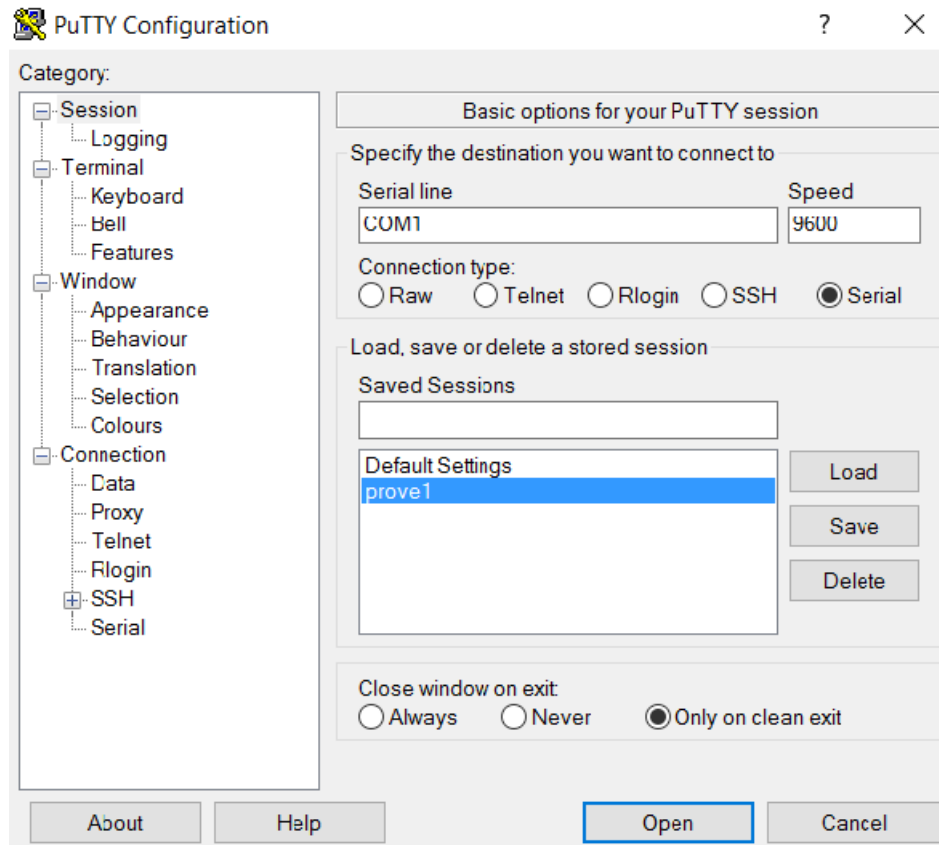


Figure 3-8. Putty connections interface. [own source]

The Evaluation Board TD 1204 contains an “AT” command shell to interconnect the EVB TD 1204 with another device that operates as host sending commands in AT mode and receiving responses to these commands.

This way AT commands can be sent using a personal computer (PC), an external microprocessor or programming the internal ARM Cortex.

I used a PC to send the AT commands.

It is useful to send commands to the EVB 1204, so after the messages are sent via SIGFOX technology and arrive to the TD servers, the data we can retrieved.

To produce enough database to analyze the type of messages to be retrieved by the library functions, it was necessary to learn how the AT commands worked.

Once inside “PuTTY” and using the TD1204 Reference Guide I sent many messages of each kind of message that could be emulated with the evaluation board.

Each message belonged to one these categories:

1. Register: Device registration to the TD platform SENSOR.

2. Event: A discrete event regarding one of the next categories
 - 2.1: Battery
 - 2.2: Temperature
 - 2.3: Switch
 - 2.4: Connection
 - 2.5: Rssi

3. Data: GPS data message with latitude, longitude, altitude, quality and satellites in view parameters. Or represents the data of a PHONE number.

4. Service: Data of either an SMS message or a TWEET message.

5. Keepalive: Signal sent by the device periodically to monitor certain parameters like voltage, temperature and frequency.

6. Raw: Hexadecimal message with no codification. It can be retrieved as raw message or as an ASCII conversion of the message.

4. Library development

4.1. Research: API, SENSOR, JSON language and VBA language

Before making the library, it was mandatory to understand what an API was (Application Program Interface). I needed to comprehend what concepts were involved, what kind of structure was presented and how the information flow worked.

An application programming interface (API) is a set of routines, protocols, and tools for building software applications. An API expresses a software component in terms of its operations, inputs, outputs, and underlying types. An API defines functionalities that are independent of their respective implementations, which allows definitions and implementations to vary without compromising each other. A good API makes it easier to develop a program by providing all the building blocks.

An API may be used for a web-based system, an operating system, a database system, computer hardware, or software library.

A simpler way to understand it, for the illiterate in the matter (as I was when I started the project) is to think about an API as all the structures (protocols, definitions, subroutines, etc.) that allow the programmer to create a program in a higher level of abstraction without worrying about the levels underneath the API.

A good metaphor using quotidian concepts for web based API is the following:

In a restaurant, a customer sitting at a table makes an order. The action of ordering food may be compared to the action of requesting information. The kitchen is where the ingredients are and the food is prepared, so it may be compared to the database. What is missing? You may ask.

The messenger is missing, the waiter or waitress that takes the order and brings it to the kitchen and after the order is processed brings the food to the customers. So an API is the messenger that gets the requests of information, gets the message to the proper destination, and when the information is ready to be delivered back, is the one that does it.

The kind of API the project will work with is a Remote API implemented in the platform SENSOR (TDNEXT SENSOR). [7]

SENSOR is a platform developed by Telecom Design that is in charge of data gathering and storage, global M2M operation management, metrics computation and business service provider through DSL's remote endpoint exposition.

According to the developer, SENSOR is designed to cope with the worst environments; as poor bandwidth, low memory or low CPU consumption. These are fitting requirements for the Internet of Things.

Sensor has two API's: (Figure 5-1.)

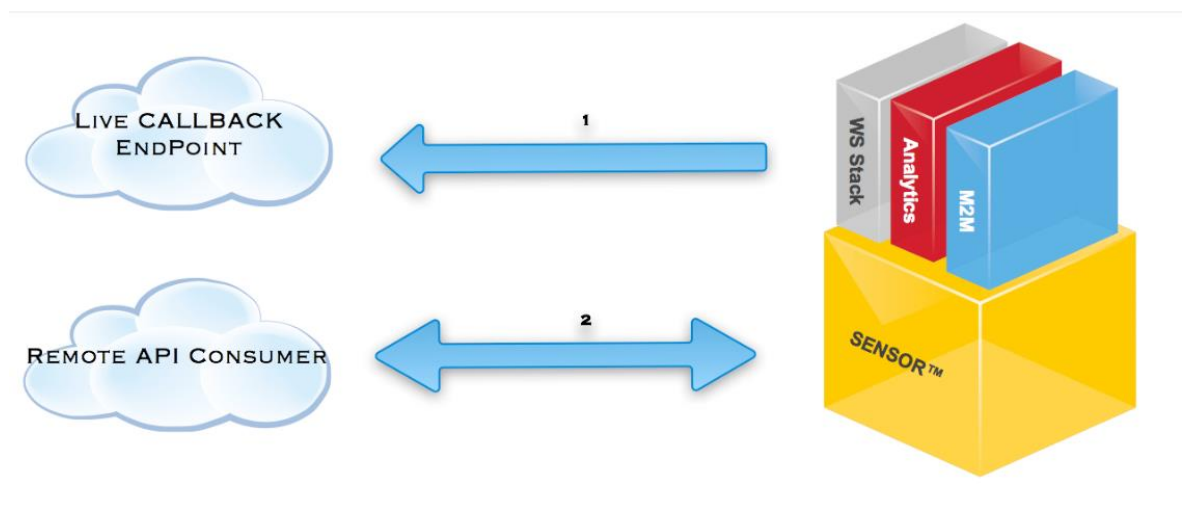


Figure 4-1. Sensor TD1202 remote Interfaces. [7]

- 1- A Live API that is related to the Callback of an event, which provides real time notification upon event broadcast. The user can either check the notifications through an on-line Dashboard or via the user's backend in case an application is configured. The next figure shows the device dashboard provided. (Figure 5-2.)

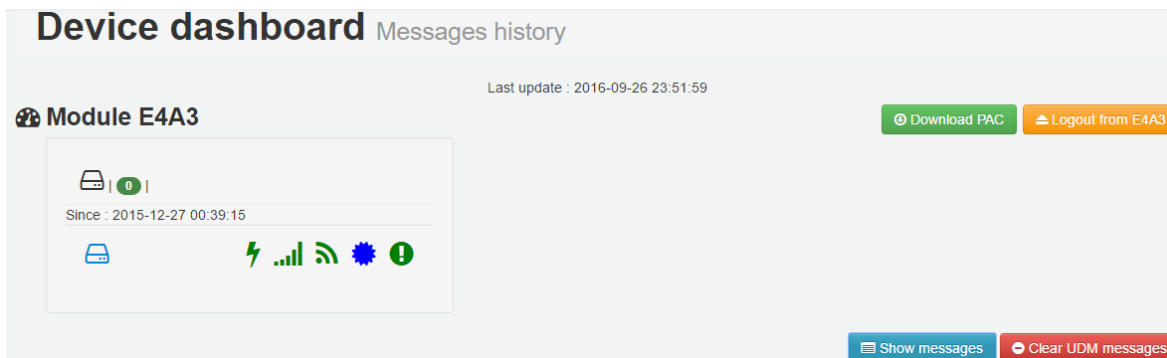


Figure 4-2 Device Dashboard overview. [7]

The information provided inside the on-line dashboard is a summary of basic parameters and basic message information.

The following figure (Figure 5-3.) shows the message information that was sent with the AT commands using the HyperTerminal program “PuTTY”.

Emission Time	Reception Time	Type	Payload	Snr (in dB)
2016-09-25 03:32:32	2016-09-25 03:32:34	raw	<ul style="list-style-type: none"> • Message : 0004474f44 • Frame Type : srv_frm_raw • Entry Id : 0 • Raw : 474f4f44 • ASCII Raw : GOOD 	13.9
2016-09-25 03:27:48	2016-09-25 03:27:51	data	<ul style="list-style-type: none"> • Message : 190101000000000000000000 • Frame Type : srv_frm_data • Type : data_gps • Entry Id : 0 • Latitude : 0 • Longitude : 0 • Altitude : 0 • Quality : 0 • Satellite In View : 0 • VIEW 	16.24
2016-09-25 03:26:25	2016-09-25 03:26:26	raw	<ul style="list-style-type: none"> • Message : 0804010203 • Frame Type : srv_frm_raw • Entry Id : 0 • Raw : 010203 • ASCII Raw : ??? 	17.49

Figure 4-3. Device API Message Information. [7]

2- A remote API that is used to retrieve and send information. The way of doing so is by making requests via SSL against the URL of the remote API.

URL: [\[https://sensor.insgroup.fr\]](https://sensor.insgroup.fr)

This remote API consists of two sub-API's. The first one is the device API [8], which can handle module operations as well as access to the stored data.

The second is the developer API which can be used to handle personal information and the applications built by the developer.

At first glance, the access to the stored data becomes the most appealing feature of the API. In particular Fetching messages history (Figure 5-4.).



GET /iot/devices/msgs/history.json

Description: Device Messages history resolution

Uri: <https://sensor.insgroup.fr/iot/devices/msgs/history.json>

Method: GET

Parameters:

1. **amount** optional the amount of event instance to be retrieved, default to 20
2. **until** optional the date (GMT timestamp) as end date, default to now

Returns: Information about last

Sample JSON response

```
[
  {
    "contrib": {
      "temp": "LOW"
    },
    "ctxt": {
      "active": true,
      "alerts": 0,
      "alerts_ack": 0,
      "battery": "UNKNOWN",
      "firstseen": "Mar 25, 2013 2:35:36 PM",
      "id": "125010855",
      "index": 6,
      "lastseen": "Mar 25, 2013 2:35:32 PM",
```

Figure 4-4. Device API. Fetching History messages detail [8]

At this point, the structure of the URL requests displayed the need to understand the language in which the information was stored. The language is JSON (JavaScript Object Notation).

At the beginning of the project I had some knowledge about XML but none of JSON. Therefore, I researched to understand the structure and behavior of this language. I used a visualization tool throughout the project that I found very useful [9].

This tool allowed me to understand the structure of the arrays and fields present within a message written in JSON.

The next figure shows the text in JSON inside the left window and the structure of the text in the right window (Figure5-5.)

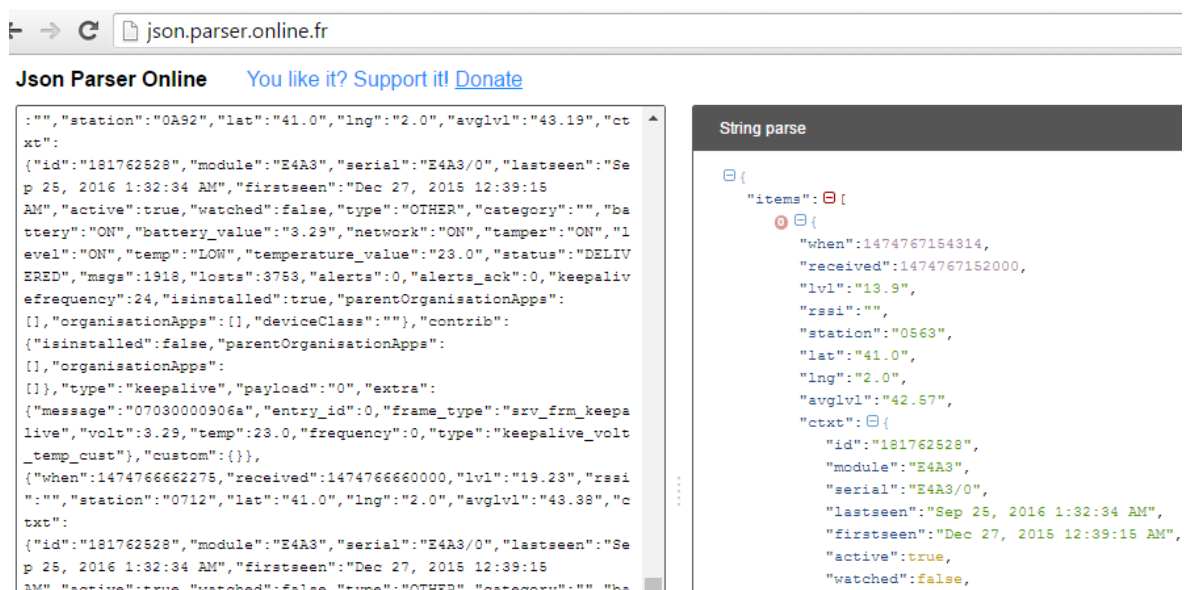


Figure 4-5. JSON parser Tool online. [9]

At this point I also researched the GET and POST methods used to make the requests [10].

This linked with the research and understanding of the language I was going to program the library with: VBA (Visual Basic for Applications).

VBA is an implementation for “Microsoft Office” programs like “Microsoft Excel” and “Microsoft Access” of the event-driven language VB (Visual Basic).

VBA enables building user-defined functions (UDFs), automating processes and accessing Windows API.

Before I was confident enough to begin writing code, I researched and practiced examples of different sources as “VBATutorialPoint” [11], MSDN VBA [12] and the content of the website of “Pearson Consulting” [13].

4.2. Desired functionality of the library.

When all the basic concepts were consolidated it was time to determine what shape the library for the Internet of Things would have and also to understand how it could be useful.

The creation of a library of functions for “Microsoft Excel” in VBA is to create a tool that helps people conciliate the relatively new and arising technologies related to the Internet of Things and the comfort of managing data with a well established program like “Microsoft Excel”.

“Microsoft Excel” is a program used mainly as a calculation tool via spreadsheets, pivot tables and macros. This means the main functionality of the library should be a way to introduce the data related to the Internet of things into “Microsoft Excel” so the user could easily use and tweak data.

Having this insight, it is only natural to focus especially on the data retrieval of the messages emitted by the Internet of Things devices, which in our case are emulated by the TD EVB 1204.

To retrieve automatically the data sent with an IoT device, that is stored at SENSOR, HTTP requests to the SENSOR site must be made. This requests are done according to the API's presented in the previous chapter 5.1.

Therefore, I proceeded to examine the list of requests that could be made using the Remote API and I figured out what functions could be made that benefited from these data requests.

The list of requests available in the Device API is: [8]

1. Authentication request:

Using the GET method and sending the ID and the Key numbers of the module, the API returns a token used as a Header for all other requests of the device.

2. Device information request. Get PAC serial number:

Using the POST method and posting the ID and Key numbers of the module, the API returns the device's PAC number.

3. Device messages request. Fetch messages history:

Using the GET method, sending the ID and Key numbers of the module and the optional parameters “number of messages to retrieve” and “date limit”, the API returns the list of messages (default to 20) stored since the first use of the module until the Date limit (default to the date and time of the request, also referred as “now”).

4. Device messages request. Fetch recent messages:

Using the GET method, sending the ID and Key numbers of the module and the optional parameters “number of messages to retrieve” and “date limit”, the API returns the list of messages (default to 20) stored since the date limit (default to now) until now.

5. Device messages request. Fetch raw messages history:

Using the GET method, sending the ID and Key numbers of the module and the optional parameters “number of messages to retrieve” and “date limit”, the API returns the list of raw messages (default to 20) stored since the date of module first use until date limit (default to now). Raw messages are messages coded by the module and not decoded.

6. Device message request. Fetching devices:

Using the GET method, sending the ID and Key numbers of the module, the API returns an Array of IoT devices registered behind a gateway module.

7. Device message request. Clear messages request:

Using the POST method and posting the ID and Key numbers of the device, clears all the messages stored in the record.

8. Device Operations. Changing device functional status:

Using the POST method and posting the ID and Key numbers and also the new device state value, updates the device state value.

9. Device Operations. Toggle device active flag:

Using the POST method and posting the ID and Key numbers and also the new active flag value, updates the device active flag value.

10. Device Operations. Toggle device monitoring flag:

Using the POST method and posting the ID and Key numbers and also the new monitoring flag value, updates the device monitoring flag value.

11. Device Operations. Change device “bidir” (bidirectional) value: Using the POST method and posting the ID and Key numbers and also the new “bidir” value, updates the device “bidir” value.

To successfully Authenticate and make an HTTP request is essential to automate the data retrieval. (adding the parameters needed depending on what is needed for each function.)

Unlike the Device API the Authentication token used for the header in the requests is not available via request itself. It can only be obtained by entering the Developer Dashboard once registration of the developer account has been created.

The next figure (Figure 5-6.) shows the developer dashboard and the Authentication token present as a String at the right of the screen.

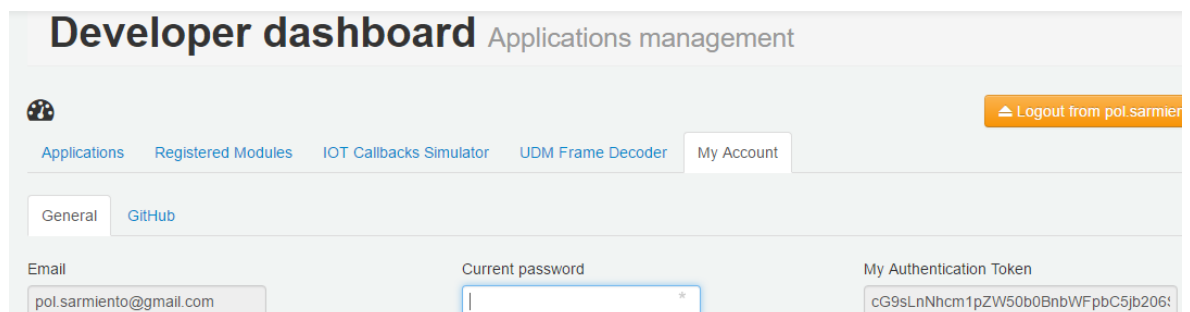


Figure 4-6. Developer Dashboard. [7]

The list of requests available in the Developer API is:

1. Developer Information. Update password:

Using the POST method by using basic HTTP authentication provided by the developer token in the developer dashboard and sending as parameters the email,

the old password, the new password and the new password again for confirmation, updates the password of the developer account.

2. Developer Information. Update GIT Alias:

Using the POST method by using basic HTTP authentication provided by the developer token in the developer dashboard and sending as parameters the email, and the developer's GIT Alias, updates the GIT Alias.

3. Developer Information. Get all your PAC serial numbers:

Using the POST method by using basic HTTP authentication provided by the developer token in the developer dashboard, it returns a text stream with all your registered modules PAC key numbers.

4. Handling Apps. IoT application listing:

Using the GET method by using basic HTTP authentication provided by the developer token in the developer dashboard, it returns the list of IoT applications in the developer account.

5. Handling Apps. IoT application creation/update:

Using the POST method by using basic HTTP authentication provided by the developer token in the developer dashboard, and sending a JSON serialized string of the application information, it creates or updates the application information created by the developer.

6. Handling Apps. IoT application deletion:

Using the DELETE method by using basic HTTP authentication provided by the developer token in the developer dashboard, and sending a JSON serialized string of the Application technical ID found in the developer dashboard, it deletes said application.

The list of requests in the developer API contains extra requests to handle specifics of the applications created with the developer functionality of SENSOR developer Dashboard.

Once the API requests have been explained it's time to discuss which of them are useful and why. Keeping in mind that the focus and main goal is to accomplish an effective and efficient data retrieval of the messages emitted by the Internet of Things devices.

The most relevant to what I'm targeting are the following from the "device API":

"1. Authentication request" and "3. Device messages request. Fetch messages history".

Those two offer the possibility to retrieve all the data of each message of an IoT device. Therefore, they will be the main way of retrieving data. It is important to note that the data is obtained as a string of JSON format and should be formatted to a more desirable way to be able to properly work with it.

Moving to the next category, the following are redundant in the purpose of retrieving the data of each message:

"2. Device information request. Get PAC serial number", "4. Device messages request. Fetch recent messages" and "5. Device messages request. Fetch raw messages history".

The PAC key is not needed for authentication if the Module Key is used, fetching messages by recent is a way of rewording the content of fetching messages by history and fetching raw messages that need to be decoded is a step back to working with clear data.

The API request "6. Device message request. Fetching devices" could be useful if a lot of devices were connected forming a net. It could be used to link data between devices and try to figure out patterns to optimize the network. However, in some applications the device is an individual with a collection of sensors which would make this request rather useless.

The following five requests could be useful pointing in another direction than data study.

7. "Device message request. Clear messages request", "8. Device Operations. Changing device functional status", "9. Toggle device active flag", "10. Device Operations. Toggle device monitoring flag" and "11. Device Operations. Change device "bidir" (bidirectional) value". All these are designed to accomplish automated operational changes of the device which is more in line with the last category. Nevertheless, they will be implemented.

Finally, the developer API requests are either for profile editing or for handling applications created by the developer to automate the behavior of the devices.

The handling and creation of applications is on a different path from the target of this project, which is more centered around data retrieval and clear data presentation.

Despite being a somewhat useful option to have these requests automated in the form of a function inside “Microsoft Excel”, the fact that the main effort of the project differs from application developing and also the need to have internet connection to check for the authentication token that is required in the developer’s API, make this functions not a priority for the library.

As a result of this analysis, from this lists of possible requests, a new list of possible functions is created. Some are directly related to a particular request and some are derived from a particular request.

The functions that will be implemented are:

(Note that are presented with the VBA header; the arguments are: “id” represents the identification module number, “key” represents the key module number, “n” represents the number of messages to be retrieved, “gmt” represents the date limit of the data retrieval, “m” represents the position of a message in a chain of “n” messages, “value” represents the new operational value for certain functions, “Adminpass” represents the administrator password to make operational changes in the device)

1. Function GetToken (id, key) As String:

This function gets the authentication token for the device API requests. This function is essential to make all the HTTP requests.

2. Function GetDataHistory (id As String, key As String, n As Integer, gmt As String) As String:

This function gets all the data available contained in the messages at SENSOR. This function is essential to retrieve full data information.

3. Function GetPayloadMessage (id, key, n As Integer, gmt As String, m As Integer) As String:

This function gets punctual payload data from a single IoT message. It allows the user to retrieve the payload of a punctual message

4. Function GetTypeMessage (id, key, n As Integer, gmt As String, m As Integer) As String:

This function gets punctual type of message data from a single IoT message. It allows the user to retrieve the type of a punctual message.

5. Function GetAsciiMessage (id, key, n As Integer, gmt As String, m As Integer) As String:

This function gets punctual ASCII message data from a single IoT message. It allows the user to retrieve the ASCII message encoded in the payload, being either a raw type message or a data type message.

6. Function GetEncodedMessage (id, key, n As Integer, gmt As String, m As Integer) As String:

This function gets punctual encoded message data from a single IoT message. It allows the user to retrieve the encoded message of a punctual message.

7. Function Get Coordinates (id, key, n As Integer, gmt As String, m As Integer) As String:

This function gets punctual coordinates data from a single IoT message. It allows the user to retrieve latitude and longitude of a single message.

8. Function GetFieldData (id, key, n As Integer, gmt As String, m As Integer) As String:

This function gets the information of the field data from a single IoT message.

9. Function GetPAC (id As String, key As String) As String:

This function gets the PAC key of the module. It can be used by the user to replace the key parameter if needed.

10. Function GetDataRecent (id As String, key As String, n As Integer, gmt As String) As String:

This function gets all the data from IoT messages by method recent.

11. Function GetChildren (id, key) As String:

This function gets the children information of a gateway module.

12. Function ClearData (id, key, Adminpass As String) As String:

This function clears data from the IoT messages database providing the admin pass first. The user can erase the database at SENSOR.

13. Function ChangeDfuncstat (id As String, key As String, value As Variant, Adminpass As String) As String

This function changes the device functional status value. It needs password.

14. Function Toggleactiveflag (id As String, key As String, value As Variant, Adminpass As String) As String:

This function toggles the active flag value of the module. It needs password.

15. Function Togglemonitorflag (id As String, key As String, value As Variant, Adminpass As String) As String:

This function changes the monitoring flag value of the module. It requires password.

16. Function ChangeBidir (id As String, key As String, value As Variant, Adminpass As String) As String:

This function changes the bidirectional value of the module. Requires password.

17. Function GetLastSeen (id, key, n As Integer, gmt As String, m As Integer) As String:

This function recovers the last date the device was operational. It requires the user to state the date limit.

18. Function GetFirstSeen (id, key, n As Integer, gmt As String, m As Integer) As String:

This function recovers the first date the device was operational.

19. Function GetMessageSent (id, key, n As Integer, gmt As String, m As Integer) As String:

This function recovers the date on which a punctual message was sent by the device.

20. Function GetMessageReceived (id, key, n As Integer, gmt As String, m As Integer) As String:

This function recovers the date on which a punctual message was received by SENSOR.

21. Function GetTravelTimeMessage (id, key, n As Integer, gmt As String, m As Integer) As String:

This function recovers the time difference between the date the message was received by SENSOR and the date the message was sent by the device. Usually, around 2 to 4 seconds.

22. Function GetCurrentMessagesServer (id, key, gmt As String) As String:

This function recovers the current number of messages stored at SENSOR.

23. Function GetTotalMessagesServer (id, key, gmt As String) As String:

This function recovers the total number of messages that have been stored at SENSOR at some point.

Some of these functions are related directly to a function from the device API, and some are derived from the functions in the API.

These functions constitute a wide range of possibilities to retrieve data that might be useful to the user.

By retrieving punctual information that might constitute an interesting value to create an application, or by retrieving full data information capable of explaining results or traits of the devices or by creating a way to interact with the device operational state.

These functions will be part of the library. This will be implemented in a module called "TDdeviceAPI".

This module will contain three processes that will be discussed in the next chapter "HttpRequest", "HttpRequest2" and "FuncDesc".

The module "TDdeviceAPI" is the part of the library where all basic actions will be represented.

4.3. Code structure and implementation

4.3.1. General considerations

Once the functionality of the library is determined by the premise of presenting usable data, some issues arise that need to be solved.

Once the first iterations of code were obtained, it was obvious that something was missing in order to present all the fields containing information that were inside the string of JSON text of the message. It was necessary to separate the information and allocate each field in a different cell.

The next figure shows the JSON string obtained by using the function GetDataHistory (Figure 5-7.).

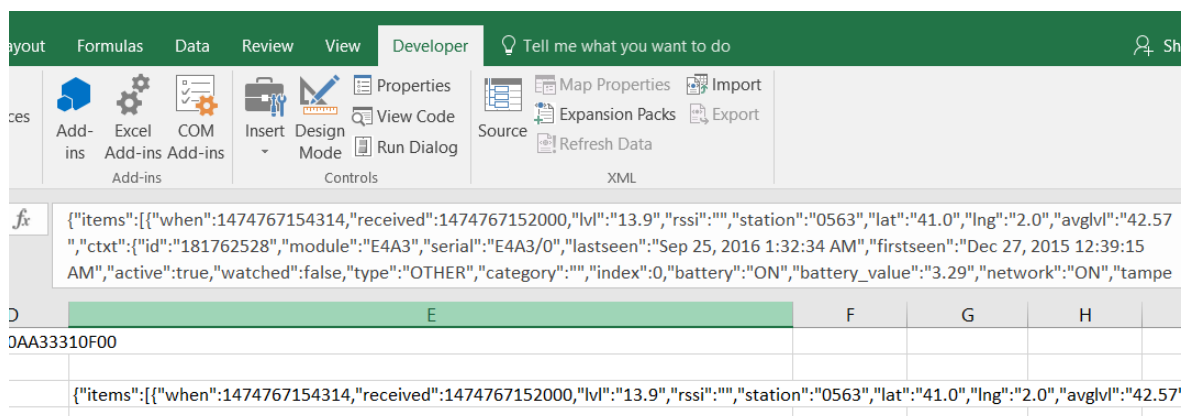


Figure 4-7. JSON string from GetDataHistory. [own source]

At this point, the idea of creating a program that could automatically generate a report was conceived. This idea will be developed and the code shown in “5.3.3 Program Structure”.

The second issue was about parsing the information in a JSON string in an effective way. It is necessary to present the information in a format where each field can be sorted and used for calculations individually.

This issue was addressed in some iterations, each iteration of the code made it better. In order to solve a problem in the code I found a piece of code that did very well the same thing I was attempting to do.



After checking the documentation and copyright I found out that it didn't impose many restrictions to use it. It stated that if significantly large sections of the code were used "As Is" it was mandatory to reproduce the copyright text at the beginning of the code.

At first I was only going to take the piece of code I needed, which was a function that counted and separated the blank spaces inside the JSON string, but I found it would be more useful to the user to have the full library available so I left it "As Is".

This library is named "JsonConverter" and the main functionality is to parse JSON string into a format that "Microsoft Excel" can accurately use. It also contains a function to convert strings of text to a JSON formatted string.

This second functionality was not used during the creation of the program, but was left inside the library in case it could be useful in the future.

The next diagram shows the structure of the information flow within the modules we have so far. 1. "TDdeviceAPI" and 2. "JsonConverter". (Figure 5-8.)

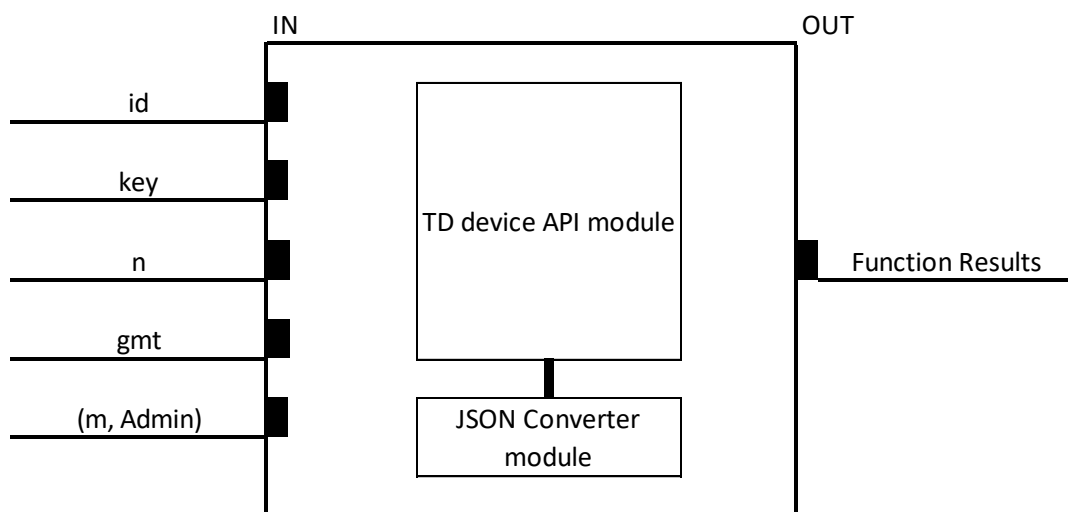


Figure 4-8. Block diagram of the first modules. [own source]

The inputs are the variables id (identification number of the module), key (key number of the module), n (number of messages to retrieve), gmt (date limit), m (position of a message within n messages) and Admin (administrator password).

The outputs are the function results of each individual function.

4.3.2. TDdeviceAPI library

The functions of “TDdeviceAPI” work by a main procedure of Http Request. An Http Request uses the HTTP protocol to gather data from an URL. The next diagram shows the structure of the HTTP Requests. (Figure 5-9.)

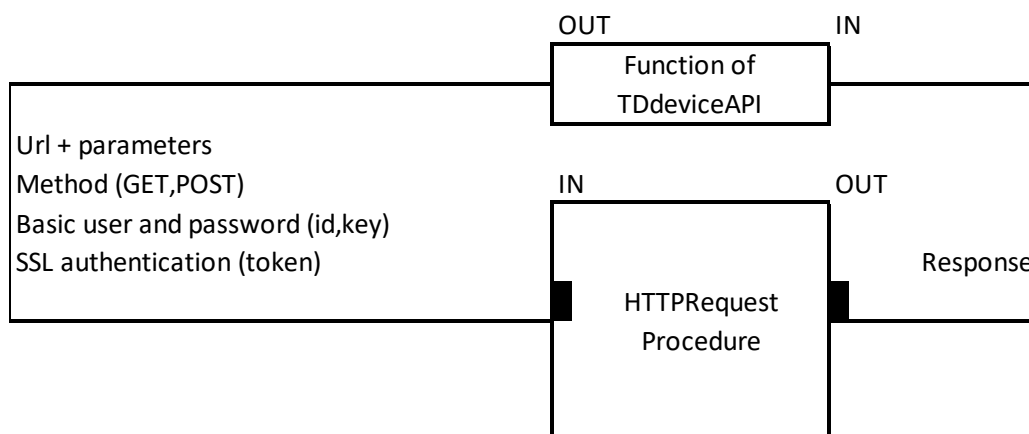


Figure 4-9. HTTP Request Procedure Structure. [own source]

I created two procedures within this library to accomplish this. The first is “HttpRequest” and the second is “HttpRequest2”. The only difference between the two is that HttpRequest2 allows an extra input, needed in some functions as “Function Toggle monitoring active flag “.

The code of “sub HttpRequest” is:

The header and local variable declaration. (Figure 5-10.)

```
Sub HttpRequest(sUrl As String, token As String, id As Variant, key As Variant, Method As String, _
  Typo As Integer, Response As String)

  Dim answer As String
  Dim loading As String
```

Figure 4-10 HTTP Request Header and variables

With a select case structure, we create an array of the possible types of Http requests. Each request is done by creating a “Microsoft.XMLHTTP” object. The methods used in the first



GET request are “.Open” to create the structure used in an artificial browser. It requires to “.SetRequestHeader” to specify the authentication token and the content type. After that the method “.Send” is used to try to establish connection with the destination. After connection is successful or unsuccessful, the fields “Response”, “answer” and “loading” are filled.

```
Select Case Typo|  
  
Case 1 'GET request with authentication and json content  
  
    With CreateObject("Microsoft.XMLHTTP")  
        .Open Method, sUrl, False, id, key  
        .SetRequestHeader "X-Snsr-Device-Key", token  
        .SetRequestHeader "Content-Type", "application/json"  
        .Send  
        Response = .responseText  
        answer = .Status  
        loading = .ReadyState  
  
    End With
```

Figure 4-11. HTTP Request GET [own source]

The following types of requests are a GET request without the need of extra headers and a POST request. In the POST request the parameters of “user” and “password” are sent inside the message (inside .Send). By comparison the parameters in the GET request are presented in the URL (inside .Open).

Case 2 'GET request without authentication

```
With CreateObject("Microsoft.XMLHTTP")
    .Open Method, sUrl, False
    .Send
    Response = .responseText
    answer = .Status
    loading = .ReadyState
End With
```

Case 3 ' POST Request with Authentication

```
With CreateObject("Microsoft.XMLHTTP")
    .Open Method, sUrl, False
    .SetRequestHeader "X-Snsr-Device-Key", token
    .Send ("sn=" + id + "&key=" + key)
    Response = .responseText
    answer = .Status
    loading = .ReadyState
End With
```

End Select

Figure 4-12. HTTP Request GET and POST [own source]

After the connection has been requested, (.Readystate) and (.Status) are checked. The variable answer is checked to ensure the request code has been successfully delivered (Ok is 200). Loading is checked to ensure no loading error has occurred. (Ok is 4).

After this, the value of the request in text format is loaded in the variable Response.

```
If loading = 4 Then

    If answer = "200" Then
        If answer = "200" Then
            Else
                Response = "Url error"
            End If
        Else
            End If

    Else
        Response = "Loading error"
    End If

End Sub
```

Figure 4-13. HTTP Request check connection [own source]

The code of a lot of functions is fairly similar. I will expose two interesting cases, “Function GetDataHistory” as an example of information related to the API request. “Function GetAsciiMessage” as an example of information derived from the API request and with the messaged parsed and separated into the categories that are desired to present.

- 1- “Function GetDataHistory”: the id, key, n and gmt values are the inputs of the function. The function first calculates the identification token with GetToken(id, key). After that the URL, method of request and type of request are initialized.

```
Function GetDataHistory(id As String, key As String, n As Integer, gmt As String) As String
    Dim Response As String
    Dim sUrl As String
    Dim Json As Object
    Dim token As String
    Dim Method As String
    Dim Typo As Integer

    token = GetToken(id, key)
    sUrl = "http://sensor.insgroup.fr/iot/devices/msgs/history.json?&amount=" + CStr(n) + "&until=" + CStr(gmt)
    Method = "GET"
    Typo = 1
```

Figure 4-14. GetDataHistory Header and variables [own source]

Next, the HttpRequest procedure is called. Then the Response is analyzed and it either returns the information as an output or shows a message box indicating what has gone wrong.

```
Call HttpRequest(sUrl, token, id, key, Method, Typo, Response)

If Response = "Url error" Then
    MsgBox ("Url Error, please try again in a few minutes")
    Exit Function

ElseIf Response = "Loading error" Then
    MsgBox ("Loading Error, please try again in a few minutes")
    Exit Function

Else
    GetDataHistory = Response

End If

End Function
```

Figure 4-15 GetDataHistory output selection [own source]

An example of the GetDataHistory response:

```
fx =GetdataHistory("E4A3";"CFE3A995";10;NOW())
D E
[{"items":[{"when":"1474766871080","received":"1474766868000","lvl":"16.24","rssi":"","station":"0A8E","lat":"41.0","lng":"2.0","avglvl":"42.66","ctxt":{"id":"181762528","module":"E4A3"},"serial":"E4A3/0","lastseen":"Sep 25, 2016 1:32:34 AM","firstseen":"Dec 27, 2015 12:39:15 AM","active":true,"watched":false,"type":"OTHER","category":"","index":0,"battery":"ON","battery_value":"3.29","network":"ON","tamper":"ON","level":"ON","temp":"LOW","temperature_value":"23.0","status":"DELIVERED","msgs":1921,"losts":3759,"alerts":0,"alert_s_ack":0,"position":{"latitude":"0","longitude":"0","altitude":"0","precision":"0","satellites":0},"keepalivefrequency":24,"isinstalled":true,"parentOrganisationApps":[],"organisationApps":[],"deviceClass":"","contrib":{"isinstalled":false,"parentOrganisationApps":[],"organisationApps":[]},"type":"raw","payload":"474f4f44","extra":{"message":"0004474f4f44"},"entry_id":0,"raw":"474f4f44","frame_type":"srv_frm_raw","asciiraw":"GOOD"},"custom":{}},{}
```

Figure 4-16. GetDataHistory Example response. [own source]

- 2- "Function GetAsciiMessage": The id, key, n, gmt values are inputs; 'm' is also an input, that represents the position in the chain of "n" messages to be represented.

First the token, URL, request method and request type are initialized. After that the HttpRequest procedure is called. Once it has a response it checks if there is any error and properly displays a message box if is the case.

```

Function GetAsciiMessage(id, key, n As Integer, gmt As String, m As Integer) As String

Dim Response As String
Dim sUrl As String
Dim Json As Object
Dim token As String
Dim Method As String
Dim Typo As Integer

token = GetToken(id, key)
sUrl = "http://sensor.insgroup.fr/iot/devices/messages/history.json?&amount=" + CStr(n) + "&until=" + CStr(gmt)
Method = "GET"
Typo = 1

Call HttpRequest(sUrl, token, id, key, Method, Typo, Response)

If Response = "Url error" Then
    MsgBox ("Url Error, please try again in a few minutes")
    Exit Function

ElseIf Response = "Loading error" Then
    MsgBox ("Loading Error, please try again in a few minutes")
    Exit Function

```

Figure 4-17 GetAsciiMessage header, variable and HTTP request [own source]

After that the JSON object is set to parse the information of the response string text. Once it is done, it continues to evaluate the type of message. (raw or service) to return the corresponding field (asciiraw or asciidata).

```

Else
    Set Json = JsonConverter.ParseJson(Response)

    If Json("items")(m)("type") = "raw" Then

        GetAsciiMessage = Json("items")(m)("extra")("asciiraw")

    ElseIf Json("items")(m)("type") = "service" Then

        GetAsciiMessage = Json("items")(m)("extra")("asciidata")

    Else
    End If

End If

End Function

```

Figure 4-18. GetAsciiMessage output selection. [own source]

An example of the function GetAsciiMessage retrieving the “Ascii” code of a raw message. (in this case the original data was a raw type data).


	<code>=GetAsciiMessage("E4A3";"CFE3A995";5;NOW());1)</code>
D	E
	GOOD

Figure 4-19. GetAsciiMessage response example. [own source]

This two examples illustrate the code developed to create the module TDdeviceAPI.

The next chapter “5.3.3 Program Structure” will discuss the code of the program implemented. The code will be contained in another module called “TDdataprocess”.

4.3.3. Program structure

This chapter discusses the structure of the program created and the functionality of it.

The program contains a set of subroutine and functions that work together to generate data reports. The reports present data on a spreadsheet with a header corresponding with each field in a message.

After that, the value of each cell is assigned depending on the header and the message field.

Next, the program performs format operations (will be explained in the next chapter “5.3.4 Report format”).

Finally, it assigns to a cell the value of the date and time the report was generated.

The program can generate two different kind of reports:

- 1- “Present Message Data”: report containing the fields that present information about the message itself, not including the fields that contain information about the device operational states and behaviors.
- 2- “Present Full Data”: report containing each field in each message (Up to 67 fields). Contains both message information and device information.

The program procedure name is “Sub PresentData”:

The header of the program with the input variables id, key, n, gmt, mode, wsname. “reporttype” is shown in (Figure 5-20).

```
Sub PresentData(id As String, key As String, n As Integer, gmt As String, wsname As String, _  
reporttype As Integer)  
  
Dim ws As Worksheet  
Dim WorkSheetExists As Boolean  
Dim response As String  
Dim responseJson As Object  
Dim rangedates As range  
Dim i As Integer  
Dim j As Integer  
Dim k As Integer
```

Figure 4-20. PresentData header and variable declaration. [own source]

The inputs id, key, n and gmt are explained in the previous chapter “5.3.2 TDdeviceAPI”.

The input “reporttype” is used to select which kind of report is generated.

The input “wsname” is used to choose the name of the worksheet where the report will be presented.

After variable declaration, a subroutine checks whether a worksheet with “wsname” name exists or not. In case it already exists, it allows the user to change the name. Finally, the new worksheet where the report is going to be presented is generated. (Figure 5-21).

```

For Each ws In ThisWorkbook.Worksheets 'check if exists wsname
    If UCase(ws.Name) = UCase(wsname) Then
        WorksheetExists = True
        Exit For
    End If
Next ws

If WorksheetExists Then 'create worksheet
    MsgBox "Worksheet name already taken, please select another name.", vbExclamation
    wsname = InputBox("Select new name")

    For Each ws In ThisWorkbook.Worksheets 'check if exists wsname
        If UCase(ws.Name) = UCase(wsname) Then
            WorksheetExists = True
            MsgBox "Please, make sure you choose an available name" _
                & vbCrLf & "for your worksheet and try again", vbCritical
            Exit Sub
        End If
    Next ws

    Sheets.Add.Name = wsname
Else
    Sheets.Add.Name = wsname
End If

Set ws = Worksheets(wsname)

```

Figure 4-21. PresentData worksheet creation. [own source]

Before gathering any data, a message box appears to indicate the user that depending on the user’s Internet connection or the amount of data in the MS Excel file, the program will require more time to finish running. This is important because while the program is running, no other activity can be performed on “Microsoft Excel”.

If the user accepts, the program starts running. First, the data is retrieved using the function GetDataHistory from the module TDdeviceAPI, after that the data is parsed using the library “JsonConverter”.

The program manages to print the correct data in each cell by going through a matrix structure. The columns of the matrix correlate to the fields of each message and the rows of the matrix correlate to each one of the messages that is presented.

After the data is parsed, the counters “i”, “j” (columns) and “k” (rows) that are used to go through the data are initialized.

Following the initialization of the counter, the program starts a loop for each message (child). (Figure 5-22.)

```
If MsgBox("Processing time may vary depending on" _  
& vbCr & "amount of data present in the workbook" _  
& vbCr & "or Internet connection." & vbCr & "Do you want to procede?", vbOKCancel  
  
    response = GetDataHistory(id, key, n, gmt)  
    Set responseJson = JsonConverter.ParseJson(response) 'data parsing  
  
    i = 1 'number of index in the json file  
    k = 1 'number of child  
    j = 1 'number of the column position in the excel grid  
  
    For k = 1 To n 'loop through childs  
  
        If ws.Cells(1, 1) = "" Then 'msg number header  
            ws.Cells(1, 1) = "msg number"  
        Else  
            End If  
        ws.Cells(k + 1, 1) = k 'adds msg number  
  
        i = 1 'restart index and column position  
        j = 2
```

Figure 4-22. PresentData Initializing. [own source]

```

If reporttype = 3 Then 'restarts special case report 3
    reporttype = 2
Else
End If

If responseJson("items")(k)("type") = "event:connectionok" And reporttype = 2 Then
    reporttype = 3
Else
End If

Do While i < 68

    If ws.Cells(1, j) = "" Then 'write header
        ws.Cells(1, j) = Fieldheader(i)
    Else
    End If

```

Figure 4-23 PresentData special case. [own source]

```

If arraylevel(i) = 1 Then 'write content
    ws.Cells(k + 1, j) = responseJson("items")(k)(Fieldname(i))
ElseIf arraylevel(i) = 2 Then
    ws.Cells(k + 1, j) = responseJson("items")(k)("ctxt")(Fieldname(i))
ElseIf arraylevel(i) = 3 Then
    ws.Cells(k + 1, j) = responseJson("items")(k)("ctxt")("position")(Fieldname(i))
ElseIf arraylevel(i) = 4 Then
    ws.Cells(k + 1, j) = responseJson("items")(k)("ctxt")(Fieldname(i))(0)
ElseIf arraylevel(i) = 5 Then
    ws.Cells(k + 1, j) = responseJson("items")(k)("contrib")(Fieldname(i))
ElseIf arraylevel(i) = 6 Then
    ws.Cells(k + 1, j) = responseJson("items")(k)("contrib")(Fieldname(i))(0)
ElseIf arraylevel(i) = 7 Then
    ws.Cells(k + 1, j) = responseJson("items")(k)(Fieldname(i))(0)
ElseIf arraylevel(i) = 8 Then
    ws.Cells(k + 1, j) = responseJson("items")(k)("extra")(Fieldname(i))
Else
    ws.Cells(k + 1, j) = "DisplayError" 'content error
    If MsgBox("Array level error", vbOKOnly) = vbOK Then
        Exit Sub
    Else
    End If
End If

```

Figure 4-24. PresentData assign values. [own source]

```

        If reporttype = 3 And j = 9 Then
            j = j + 26
        Else
            j = j + 1 'increase column
        End If

        i = nextindex(i, reporttype) 'increase index with desired nextindex function

    Loop

Next

ws.Cells(n + 3, 1) = "Report Created " & Now() 'report creation time

'format changes
Call autosize

Call formatcellstimedate(ws, rangedates, n)

    If MsgBox("Report Ready", vbOKOnly + vbInformation, "Report Information") = vbOK Then
    Else
    End If
Else
    Call Deleteworksheet(wsname)
    Exit Sub
End If
|
End Sub

```

Figure 4-25. PresentData restart counters and format changes. [own source]

The first part of the loop is creating the field “msg number” that indicates the order of messages of the report. (Figure 5-22.)

After that, the program evaluates if the message belongs to a problematic type (“event: connection”), which may create display issues in the report because it has information contained in different fields than the other type of messages. In case the message belongs to this type, a temporal report type (3) is assigned to it. (Figure 5-23.)

Next, for each field (i) in the message, the array depth is checked and the correct information is assigned to the correct cell. (Figure 5-24.)

After that, the counters advance (Figure 5-25.) and when the first message is completely printed on the cells of the worksheet, it goes through the matrix again (Figure 5-22.)

When all the information of every field of every message is printed on the worksheet, the date of the report generation is created beneath the last message. (Figure 5-25.)

The final stage of the report generation program is to change the format of the cells, adjusting the width of the columns and transforming data types. (Figure 5-25.)

4.3.4. Report format

In order to improve the data presentation of the report, it is necessary to adjust the width of the columns. To solve this, a process named “autosize” is called. (Figure 5-25.)

An additional display issue has to be solved, the fields “when” and “received” of the report present the information as an Epoch timestamp by default.

The Unix epoch (or Unix time or POSIX time or Unix timestamp) is the number of seconds that have elapsed since January 1, 1970 (midnight UTC/GMT), not counting leap seconds (in ISO 8601: 1970-01-01T00:00:00Z). Literally speaking the epoch is Unix time 0 (midnight 1/1/1970), but 'epoch' is often used as a synonym for 'Unix time'. Many Unix systems store epoch dates as a signed 32-bit integer, which might cause problems on January 19, 2038 (known as the Year 2038 problem or Y2038). [14]

To solve this problem two functions that transform Epoch timestamp to date format were created. (Figure 5-26.) These functions transform the Epoch timestamp to MS Excel timestamp, which is the number of seconds that have elapsed since January 1, 1900 (midnight UTC/GMT).

```
'This function transforms epoch in milliseconds timestamp to excel date
Function MilistoDate(myMillis As Double) 'transforms epoch in milliseconds timestamp to excel date
MilistoDate = (myMillis / 86400000) + 25569
'divides the epoch timestamp by the number of milliseconds in a day and adds the days that passed
'from 1900, January 1st at 00:00 to 1970, January 1st at 00:00

End Function
```

```
'This function transforms epoch timestamp to excel date
Function EpochToDate(myEpoch As Double) 'transforms epoch in seconds timestamp to excel date
EpochToDate = (myEpoch / 86400) + 25569
'divides the epoch timestamp by the number of seconds in a day and adds the days that passed
'from 1900, January 1st at 00:00 to 1970, January 1st at 00:00

End Function
```

Figure 4-26. Epoch to Date functions [own source]

The input of the first function is the epoch timestamp in milliseconds and the input of the second function is the epoch timestamp in seconds.

4.3.5. User interface

The user interface developed is based on a message box system that tells the user what parameters are needed in order to run the program.

The user is asked to enter the values of the parameters using Input boxes.

The data the user enters the program is one of the most problematic points of the program, because in case the type of data does not match the expected data in the code, the program may crash.

To avoid this issues, a lot of error handling and testing was done. (Figure 5-27. To Figure 5-31.)

```
'this sub allows a user guided process of generating reports for IoT messages

Sub Main() 'main procedure for users that wish to not use the userform
Dim id As String 'module ID number
Dim key As String 'module key number|
Dim n As Integer 'number of messages to be retrieved
Dim intn As Variant
Dim gmt As String 'date limit to be established
Dim wsname As String 'name of the report sheet
Dim reporttype As Integer 'type of report
Dim intreporttype As Variant
Dim starterexpression As String
Dim quotationmark As String

quotationmark = Chr$(34) ' Ascii for quotation mark

starterexpression = "To start the report generation program, six values are needed:" _
& vbCr & "1.The module ID number" & vbCr & "2.The module Key number" _
& vbCr & "3.The number of messages to retrieve" _
& vbCr & "4.The date limit" & vbCr & "5.The name of the worksheet" _
& vbCr & "6.The type of report" & vbCr & "Do you want to procede?" _

If MsgBox(starterexpression, vbOKCancel) = vbOK Then 'guided start

    id = InputBox("Please insert module ID number" & vbCr & "without using quotation marks." _
& vbCr & "e.g. E4A3", "1. Insert Module ID", "E4A3")
```

Figure 4-27. User interface 1. [own source]


```

Object
'error handling
If id = "" Then
    MsgBox ("You entered some invalid value!" & vbCrLf & "Please try again")
    Exit Sub
ElseIf InStr(id, quotationmark) > 0 Then
    MsgBox ("You entered some invalid value!" & vbCrLf & "Please try again")
    Exit Sub
Else
End If

key = InputBox("Please insert module Key number" & vbCrLf & "without using quotation marks." _
    & vbCrLf & "e.g. CFE3A995", "2. Insert Module Key", "CFE3A995")

'error handling

If key = "" Then
    MsgBox ("You entered some invalid value!" & vbCrLf & "Please try again")
    Exit Sub
ElseIf InStr(key, quotationmark) > 0 Then
    MsgBox ("You entered some invalid value!" & vbCrLf & "Please try again")
    Exit Sub
Else
End If

```

Figure 4-28. User interface 2. [own source]

```

intn = InputBox("How many messages do you want to retrieve?", "3. Number of messages")

'error handling

If Not IsNumeric(intn) Then
    MsgBox ("You entered some invalid value!" & vbCrLf & "Please try again")
    Exit Sub
ElseIf Not Round(intn, 0) / intn = 1 Then
    MsgBox ("You entered some invalid value!" & vbCrLf & "Please try again")
    Exit Sub
Else
    n = CInt(intn)
End If

Select Case n
    Case 0
        MsgBox ("You entered some invalid value!" & vbCrLf & "Please try again")
        Exit Sub
    Case Else
End Select

```

Figure 4-29. User interface 3. [own source]

```

If gmt = "" Then
    MsgBox ("You entered some invalid value!" & vbCrLf & "Please try again")
    Exit Sub
ElseIf InStr(gmt, quotationmark) > 0 Then
    MsgBox ("You entered some invalid value!" & vbCrLf & "Please try again")
    Exit Sub
Else
End If

wsname = InputBox("How do you want to name the worksheet?" _
& vbCrLf & "e.g. Report1", "5. Destination Worksheet name")

'error handling

If wsname = "" Then
    MsgBox ("You entered some invalid value!" & vbCrLf & "Please try again")
    Exit Sub
ElseIf InStr(wsname, quotationmark) > 0 Then
    MsgBox ("You entered some invalid value!" & vbCrLf & "Please try again")
    Exit Sub
Else
End If

```

Figure 4-30. User interface 4. [own source]

```

intreporttype = InputBox("Thank you. If you want to Present Message Data insert 1." _
& vbCrLf & "If you want to Present Full Data insert 2", "6. Report type")

'error handling

If Not IsNumeric(intreporttype) Then
    MsgBox ("You entered some invalid value!" & vbCrLf & "Please try again")
    Exit Sub
Else
    reporttype = CInt(intreporttype)
End If

Select Case reporttype
    Case 1
    Case 2
    Case Else
        MsgBox ("You entered some invalid value!" & vbCrLf & "Please try again")
        Exit Sub
End Select

Call PresentData(id, key, n, gmt, wsname, reporttype)
Exit Sub
Else
    Exit Sub
End If

End Sub

```

Figure 4-31. User interface 5. [own source]

4.3.6. TDfunctionsDescriptions

The last of the four modules developed, is created to help the user understand what the functions built in the module TDdeviceAPI do.

The module uses “Application.MacroOptions” to incorporate the function description and the function arguments description to the “Insert function” tab in MS Excel.

For example: (Figure 5-32. And Figure 5-33.)

```
'get ascii message info

Private Sub DGetAsciiMessage()
Dim fname As String
Dim fdescrip As String
Dim fcateg As String
Dim argdescrip(1 To 5) As String

fname = "GetAsciiMessage"
fdescrip = "Obtains the Ascii conversion of the 'm' message of a list of last 'n' messages"
fcateg = "TD IoT"
argdescrip(1) = "Module's ID number inside brackets."
argdescrip(2) = "Module's Key number inside brackets."
argdescrip(3) = "Number of messages to retrieve. E.g.: 1"
argdescrip(4) = "Excel timestamp of GMT time in 'dd/mm/yyyy hh:mm:ss' format E.g.: 18/08/2016 00:00:00 "
argdescrip(5) = "The number of the message you want to show. E.g.: 1"

Application.MacroOptions Macro:=fname, Description:=fdescrip, Category:=fcateg, ArgumentDescriptions:=arg
End Sub
```

Figure 4-32. Description GetAsciiMessage. [own source]

```
'get total number messages on server message info

Private Sub DGetTotalMessagesServer()
Dim fname As String
Dim fdescrip As String
Dim fcateg As String
Dim argdescrip(1 To 3) As String

fname = "GetTotalMessagesServer"
fdescrip = "Obtains the total number of messages sent to the server with this device"
fcateg = "TD IoT"
argdescrip(1) = "Module's ID number inside brackets."
argdescrip(2) = "Module's Key number inside brackets."
argdescrip(3) = "Excel timestamp of GMT time in 'dd/mm/yyyy hh:mm:ss' format E.g.: 18/08/2016 00:00:00 "

Application.MacroOptions Macro:=fname, Description:=fdescrip, Category:=fcateg, ArgumentDescriptions:=arg
End Sub
```

Figure 4-33. Description GetTotalMessagesServer. [own source]

4.4. Library diagram

The final library consists of the four modules explained in the previous chapters. These modules are:

- 1- TDdeviceAPI: includes functions that use HTTP requests to retrieve data from SENSOR (punctual data or full data) and the processes used to make HTTP requests.
- 2- JsonConverter: auxiliary module used to parse JSON messages.
- 3- TDdataprocess: includes functions and processes that create the report generation program.
- 4- TDfunctionsDescriptions: includes functions and processes to help the user.

These libraries are meant to be used together. But the user may choose to only use part of them.

“TDdeviceAPI” is the cornerstone of the library.

“JsonConverter” is used in some functions of “TDdeviceAPI” and in the main process of “TDdataprocess”.

“TDdataprocess” requires “TDdeviceAPI” and “JsonConverter” to be operational.

“TDfunctionsDescriptions” is additional content independent of the other modules.

The following diagram shows how the modules that conform the library are interconnected. (Figure 5-34.)

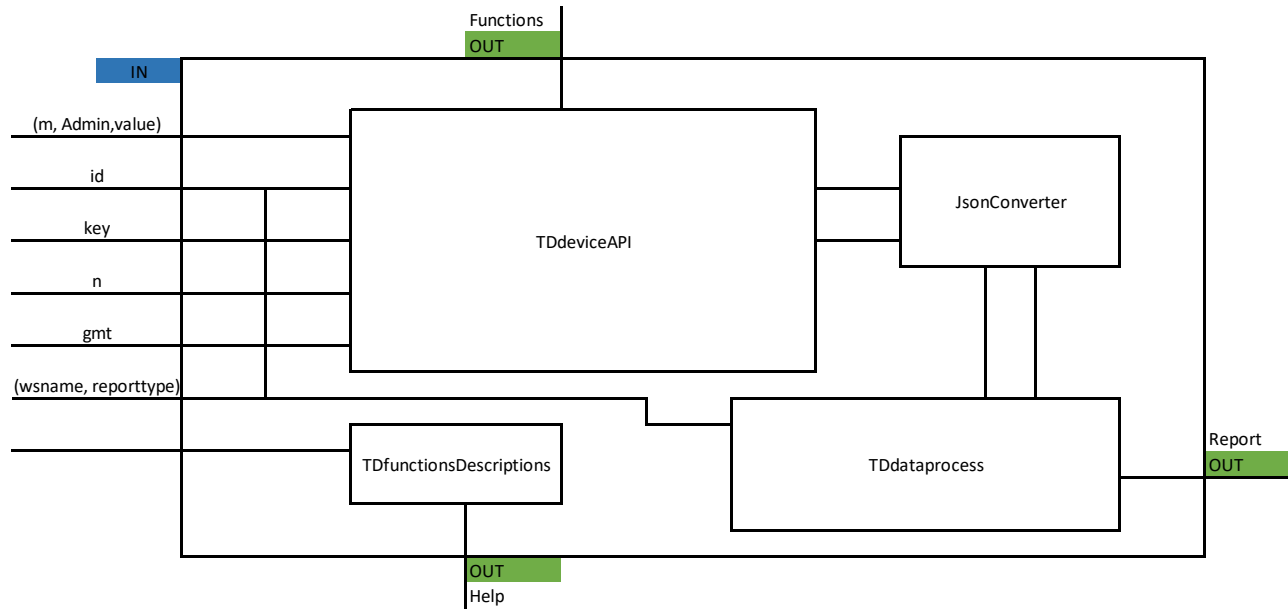


Figure 4-34. Library Diagram. [own source]

5. Test and validation

To validate the use of the library, tests were conducted on the functions and processes of the library to determine their functionality.

5.1. TDdeviceAPI functions

The functions tested were all the functions contained in TDdeviceAPI module. Tests changing each of the input parameters of each function were conducted.

The results of a sample test are shown in the following tables (Figure 6-1. and Figure 6-2.)

Function	Result	Expected
GetToken("E4A3";"CFE3A995")	MDAwMEU0QTM6Q0ZFM0E5OTU=	Yes
GetPAC("E4A3";"CFE3A995")	FOE820AA33310F00	Yes
GetDataHistory("E4A3";"CFE3A995";5;NOW())	{"items":[{"when":1474767154314,"received":1474767152000,"lvl":13.9,"rssi":"","station":0563	Yes
GetChildren("E4A3";"CFE3A995")	[{"id":181762528,"module":E4A3,"serial":E4A3/0,"lastseen":Sep 25, 2016 1:32:34 AM,"first	Yes
GetCoordinates("E4A3";"CFE3A995";1;NOW();1)	0°N 0°E	Yes
GetAsciiMessage("E4A3";"CFE3A995";1;NOW();1)	GOOD	Yes
GetFirstSeen("E4A3";"CFE3A995";1;NOW();1)	Dec 27, 2015 12:39:15 AM	Yes
GetLastSeen("E4A3";"CFE3A995";1;NOW();1)	Sep 25, 2016 1:32:34 AM	Yes
GetMessageReceived("E4A3";"CFE3A995";1;NOW();1)	09/25/16 01:32:34	Yes
GetMessageSent("E4A3";"CFE3A995";1;NOW();1)	09/25/16 01:32:32	Yes
GetTravelTimeMessage("E4A3";"CFE3A995";1;NOW();1)	00:00:02	Yes

Figure 5-1. Function Test Sample1. [own source]

GetCurrentMessagesServer("E4A3";"CFE3A995";NOW())	59	Yes
GetEncodedMessage("E4A3";"CFE3A995";1;NOW();1)	0004474f4f44	Yes
GetFieldData("E4A3";"CFE3A995";1;NOW();1)		Yes
GetPayloadMessage("E4A3";"CFE3A995";1;NOW();1)	474f4f44	Yes
GetTypeMessage("E4A3";"CFE3A995";1;NOW();1)	raw	Yes

Figure 5-2. Function Test Sample2. [own source]

The sample tests of the functions that change the operational status of the device are not shown because the output cannot be seen through screen.

msg number	when	received	lvl	rss	station	lat	lng	avglvl	ctxt.id	ctxt.module	ctxt.serial	ctxt.lastseen	ctxt.firstsi
1	09/25/16 01:32:34	09/25/16 01:32:32	13,9		563	41	2	42,57	181762528	E4A3	E4A3/0	25/09/2016 1:32	27/12/20
2	09/25/16 01:27:51	09/25/16 01:27:48	16,24		0A8E	41	2	42,66	181762528	E4A3	E4A3/0	25/09/2016 1:32	27/12/20
3	09/25/16 01:26:27	09/25/16 01:26:25	17,49		563	41	2	42,73	181762528	E4A3	E4A3/0	25/09/2016 1:32	27/12/20
4	09/25/16 01:24:41	09/25/16 01:24:39	16,9		0A92	41	2	43,19	181762528	E4A3	E4A3/0	25/09/2016 1:32	27/12/20
5	09/25/16 01:24:22	09/25/16 01:24:20	19,23		712	41	2	43,38	181762528	E4A3	E4A3/0	25/09/2016 1:32	27/12/20
6	09/25/16 01:24:04	09/25/16 01:24:02	18,53		0D60	41	2	43,67	181762528	E4A3	E4A3/0	25/09/2016 1:32	27/12/20
7	09/25/16 01:23:46	09/25/16 01:23:44	13,4		670	41	2	43,53	181762528	E4A3	E4A3/0	25/09/2016 1:32	27/12/20
8	09/25/16 01:22:19	09/25/16 01:22:17	10,73		670	41	2	43,61	181762528	E4A3	E4A3/0	25/09/2016 1:32	27/12/20
9	09/25/16 01:21:48	09/25/16 01:21:45	19,33		712	41	2	43,87					
10	09/25/16 01:20:42	09/25/16 01:20:39	8,62		08A7	41	2	45,26	181762528	E4A3	E4A3/0	25/09/2016 1:32	27/12/20
11	09/25/16 01:20:01	09/25/16 01:19:59	17,49		563	41	2	51,01	181762528	E4A3	E4A3/0	25/09/2016 1:32	27/12/20
12	09/18/16 00:18:50	09/18/16 00:18:47	16,32		563	41	2	50,97	181762528	E4A3	E4A3/0	25/09/2016 1:32	27/12/20
13	09/18/16 00:17:46	09/18/16 00:17:44	14,43		1F76	41	2	50,95	181762528	E4A3	E4A3/0	25/09/2016 1:32	27/12/20
14	09/18/16 00:17:09	09/18/16 00:17:06	18,86		563	41	2	54,84	181762528	E4A3	E4A3/0	25/09/2016 1:32	27/12/20
15	09/18/16 00:16:38	09/18/16 00:16:36	11,75		090B	41	2	56,86	181762528	E4A3	E4A3/0	25/09/2016 1:32	27/12/20

Figure 5-5. Full Report Test Sample 1. [own source]

15	09/18/16 00:16:38	09/18/16 00:16:36	11,75		090B	41	2	56,86	181762528	E4A3	E4A3/0	25/09/2016 1:32	27/12/201
16	09/17/16 23:27:24	09/17/16 23:27:22	6,95		090B	41	2	56,86	181762528	E4A3	E4A3/0	25/09/2016 1:32	27/12/201
17	09/17/16 23:22:59	09/17/16 23:22:57	17,09		563	41	2	56,85	181762528	E4A3	E4A3/0	25/09/2016 1:32	27/12/201
18	09/17/16 23:16:04	09/17/16 23:16:02	17,1		563	41	2	56,84	181762528	E4A3	E4A3/0	25/09/2016 1:32	27/12/201
19	09/17/16 23:08:14	09/17/16 23:08:12	9,08		579	41	2	56,82	181762528	E4A3	E4A3/0	25/09/2016 1:32	27/12/201
20	09/17/16 23:00:22	09/17/16 23:00:20	19,9		556	41	2	56,82	181762528	E4A3	E4A3/0	25/09/2016 1:32	27/12/201
Report Created 14/10/2016 7:34:03													

Figure 5-6. Full Report Test Sample 2. [own source]



5.3. User interface parameters

To avoid problems related to the data entered by the user, tests of every input parameter were conducted.

Tests were conducted in which misspelling errors were forced to find a solution of the issues that may be encountered.

The Error handling done can be seen in the figures (5-27. To 5-31.)

5.4. Miscellanea

The tests in the previous chapters (6.1., 6.2. and 6.3.) were conducted in MS Windows x64 systems and MS Windows x32 systems. Both systems supported the library and the results were the same in each type of Windows system.

Tests using incorrect URL's and parameters were conducted to check the outcome in case of server failure. The errors that were displayed in screen were expect.

Speed tests of the report generation program were conducted under different conditions (different Internet connection speeds and different volume of data in the spreadsheet), The results were the same. Under optimal conditions, the program performed correctly and quickly.

6. Budget

The work costs include all the money spent in the student during the months of work.

The Budget of this Project consists of three components:

1. The cost of the hardware and software used to develop the library:

Hardware and Software			
Item	Amount	Unitary price(€/uni)	Total price (€)
Laptop amortization	0,1	800	80
Microsoft Office Subscription	6	10	60
EVB TD 1204	1	180	180
TOTAL			320

2. The cost of the hours worked on the project:

Engineer worked time			
Concept	Hours	Revenue (€/h)	Total price (€)
Engineering work	500	40	20000

3. The cost associated with the consumption of resources

Resource consumption			
Concept	Hours	Price	Total price (€)
Energy consumption (100W)	500	0,22 (€/kWh)	11
Internet	5 month	40 (€/month)	200
TOTAL			211

The Budget of the project is 20531 €.

7. Environmental Impact

This project's environmental impact is minimal due to the fact it was fully developed working with a PC and EVB TD 1204.

Therefore, neither the making of project, or the results obtained by it, produce any additional environmental impact regarding fumes, waste and worn out tools.

Even so, some materials were used during the realization of this project, such as office material, that do bear a significant environmental impact during their production.

Moreover, the waste generated through the transportation of this materials to their destination and the recyclability and reusability of the components of this materials should be taken into consideration when considering the environmental impact of this project.

Finally, we must consider the radio electric contamination derived from the messages through the TD1204 module, which, in this project and according to the current Spanish legislation on radio electric spectrum, operates inside acceptable parameters.

Conclusions

The objectives defined during the project were to create an Excel VBA library for the Internet of Things.

This library had to be solid, robust and focused on presenting the data retrieved clearly and in a way that it would allow the library's user to work easily.

The code is refined and the tests of the functions and reports were successful.

The final goal of this project is to provide a tool for users of devices belonging to the Internet of Things, as well as to provide a framework to develop new projects related to the IoT was accomplished.

Acknowledgments

I would like to thank my project supervisor Juan Manuel Moreno for helping and advising me when I most needed.

I would also thank Carme Lozano Farriol, Patrícia Sarmiento Lozano and Joan Sarmiento Bueno to all their support throughout the years.

Finally, I would like to thank Claudia Sisquellas Hernández for all the support and advice that gave me when I was stuck at some point.

Bibliography

- [1]** CUJO, Why aren't we worried about the Internet of Things.
[<https://www.getcujo.com/internet-of-things-security-device-cujo/>, September 25, 2016]
- [2]** LARRY DIGNAN, Internet of things: Poised to be a security headache?, Between the Lines, November 18, 2014
- [3]** PETER R. EGLI, Overview of Emerging Technologies for low power wide area networks in internet of things and M2M scenarios. LPWAN (Low Power Wide Area Network).
- [4]** SHIKAI ZHANG, UNB modulation salvages spectrum, April 16, 2009
[<http://mwrf.com/markets/unb-modulation-salvages-spectrum>]
- [5]** SIGFOX, About SIGFOX
[<http://makers.sigfox.com/>]
- [6]** TDNEXT, Evaluation Boards, November 2015.
[<http://rfmodules.td-next.com/modules/>]
- [7]** TDNEXT, SENSOR platform, November 2015.
[<https://developers.insgroup.fr/iot/about.html>]
- [8]** TDNEXT, SENSOR device API, November 2015.
[<https://developers.insgroup.fr/iot/device.html>]
- [9]** ALAIN CUENOT, JSON parser tool online
[<http://json.parser.online.fr/>]
- [10]** MICROSOFT DEVELOPER NETWORK, HTTP Requests.
[[https://msdn.microsoft.com/en-us/library/system.net.httpwebrequest.method\(v=vs.110\).aspx](https://msdn.microsoft.com/en-us/library/system.net.httpwebrequest.method(v=vs.110).aspx)]

[11] TUTORIALSPPOINT, VBA (Visual Basic for Applications).

[<https://www.tutorialspoint.com/vba/>]

[12] MICROSOFT DEVELOPER NETWORK, VBA (Visual Basic for Applications).

[[https://msdn.microsoft.com/en-us/library/office/ee814735\(v=office.14\).aspx#VBA Programming 101](https://msdn.microsoft.com/en-us/library/office/ee814735(v=office.14).aspx#VBAProgramming101)]

[13] CPEARSON, VBA examples.

[<http://www.cpearson.com/Excel/MainPage.aspx>]

[14] MISJA.COM, Epoch Converter. 2016

[<http://www.epochcoverter.com/>]