BRNO UNIVERSITY OF TECHNOLOGY

Faculty of Electrical Engineering
and Communication

MASTER'S THESIS

Brno, 2018 Bc. David Hudec

BRNO UNIVERSITY OF TECHNOLOGY

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

FAKULTA ELEKTROTECHNIKY
A KOMUNIKAČNÍCH TECHNOLOGIÍ

DEPARTMENT OF TELECOMMUNICATIONS

ÚSTAV TELEKOMUNIKACÍ

IMPLEMENTATION OF M2M DATA GENERATOR UTILIZING COMMUNICATION PROTOCOL WIRELESS M-BUS IN SMART GRID INFRASTRUCTURE

GENERÁTOR M2M DAT BEZDRÁTOVÉHO PROTOKOLU WIRELESS M-BUS V SMARTGRID

MASTER'S THESIS
DIPLOMOVÁ PRÁCE

AUTHOR — **Bc. David Hudec**
AUTOR PRÁCE

SUPERVISOR — **Ing. Pavel Mašek, Ph.D.**
VEDOUCÍ PRÁCE

BRNO 2018

BRNO UNIVERSITY OF TECHNOLOGY
FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

Master's Thesis

Master's study field **Communications and Informatics**
Department of Telecommunications

Student: Bc. David Hudec ***ID:*** 165289

Year of study: 2 ***Academic year:*** 2017/18

TITLE OF THESIS:

Implementation of M2M Data Generator Utilizing Communication Protocol Wireless M-BUS in Smart Grid Infrastructure

INSTRUCTION:

The diploma thesis addresses the utilization of the communication protocol Wireless M-BUS as the communication container for Machine-to-Machine (M2M) data communications within the Smart Grid networks. The theoretical part of the thesis will consist of the thorough description of the WM-BUS communication protocol i.e., security aspects, communication modes, data structure, etc. At the length of the practical part, the attention will be focused on the design and implementation of the multi-platform M2M data generator utilizing the WM-BUS protocol. The key output of the thesis will be the application (controlled via console and graphical user interface, respectively) for generating WM-BUS data toward the data concentrator in Smart Grid infrastructure.

RECOMMENDED LITERATURE:

[1] BOSWARTHICK, David, Omar ELLOUMI a Olivier HERSENT. 2012. M2M communications: a systems approach. Hoboken, N.J.: Wiley, xxiii, 308 p. ISBN 978-1-119-99475-6.

[2] HERSENT, Olivier., David. BOSWARTHICK a Omar. ELLOUMI, 2012. The internet of things: key applications and protocols. Chichester, West Sussex: Wiley. ISBN 978-1119994350.

Date of project specification: 5.2.2018 ***Deadline for submission:*** 21.5.2018

Leader: Ing. Pavel Mašek, Ph.D.
Consultant:

prof. Ing. Jiří Mišurec, CSc.
Subject Council chairman

ABSTRACT

Presented thesis describes the communication protocol of Wireless M-Bus, especially its data contents, field structure, communication modes and other data-link layer and application layer details. Based on this research, a Java application designed to serve as a data generator for the protocol in question was created. Via both graphical user interface and command line interface, this program allows the user to define Wireless M-Bus telegrams in high detail and schedule them for periodic transmission via a supported hardware device to the Wireless M-Bus network. Two complete appliances, using either a standalone IQRF module or a complete UniPi Neuron control board, were developed, described and tested.

KEYWORDS

Data generator, IQRF, Java, jSSC, M2M, M-Bus, serial port communication, UniPi, UniPi Neuron, Wireless M-Bus

ABSTRAKT

V rámci této práce byl vytvořen popis bezdrátového komunikačního protokolu Wireless M-Bus, zaměřující se zejména na datovou část tohoto protokolu, strukturu jeho informačních polí, režimy komunikace a další specifika linkové a aplikační vrstvy. Na základě tohoto výzkumu byl vytvořen softwarový nástroj v jazyce Java, sloužící jako generátor dat zmíněného protokolu. Pomocí grafického i textového uživatelského rozhraní program umožňuje uživateli vytvořit Wireless M-Bus telegramy s velmi vysokou úrovní detailu a ty následně s využitím některého z podporovaných hardwarových zařízení periodicky odesílat do Wireless M-Bus sítě. Dále byla navržena dvě kompletní řešení, využívající buď samotného bezdrátového IQRF modulu nebo jeho spojení s řídicí deskou UniPi Neuron. Oba návrhy byly zrealizovány, otestovány a jsou v práci detailně popsány.

KLÍČOVÁ SLOVA

Generátor dat, IQRF, Java, jSSC, komunikace sériovým portem, M2M, M-Bus, UniPi, UniPi Neuron, Wireless M-Bus

HUDEC, David. *Implementation of M2M Data Generator Utilizing Communication Protocol Wireless M-BUS in Smart Grid Infrastructure* . Brno, 2018, 78 p. Master's Thesis. Brno University of Technology, Faculty of Electrical Engineering and Communication, Department of Telecommunications. Advised by Ing. Pavel Mašek, Ph.D.

DECLARATION

I declare that I have written the Master's Thesis titled "Implementation of M2M Data Generator Utilizing Communication Protocol Wireless M-BUS in Smart Grid Infrastructure " independently, under the guidance of the advisor and using exclusively the technical references and other sources of information cited in the thesis and listed in the comprehensive bibliography at the end of the thesis.

As the author I furthermore declare that, with respect to the creation of this Master's Thesis, I have not infringed any copyright or violated anyone's personal and/or ownership rights. In this context, I am fully aware of the consequences of breaking Regulation § 11 of the Copyright Act No. 121/2000 Coll. of the Czech Republic, as amended, and of any breach of rights related to intellectual property or introduced within amendments to relevant Acts such as the Intellectual Property Act or the Criminal Code, Act No. 40/2009 Coll., Section 2, Head VI, Part 4.

Brno ....

author's signature

Faculty of Electrical Engineering
and Communication
Brno University of Technology
Purkynova 118, CZ-61200 Brno
Czech Republic
http://www.six.feec.vutbr.cz

ACKNOWLEDGEMENT

Research described in this Master's Thesis has been implemented in the laboratories supported by the SIX project; reg. no. CZ.1.05/2.1.00/03.0072, operational program Výzkum a vývoj pro inovace.

Brno ....
author's signature

EVROPSKÁ UNIE
EVROPSKÝ FOND PRO REGIONÁLNÍ ROZVOJ
INVESTICE DO VAŠÍ BUDOUCNOSTI

Tato práce vznikla jako součást klíčové aktivity KA6 - Individuální výuka a zapojení studentů bakalářských a magisterských studijních programů do výzkumu v rámci projektu OP VVV Vytvoření double-degree doktorského studijního programu Elektronika a informační technologie a vytvoření doktorského studijního programu Informační bezpečnost, reg. č. CZ.02.2.69/0.0/0.0/16_018/0002575.

EVROPSKÁ UNIE
Evropské strukturální a investiční fondy
Operační program Výzkum, vývoj a vzdělávání

MINISTERSTVO ŠKOLSTVÍ, MLÁDEŽE A TĚLOVÝCHOVY

Projekt je spolufinancován Evropskou unií.

ACKNOWLEDGEMENT

I would like to thank my supervisor, Ing. Pavel Mašek, Ph.D., for the professional leading, consulting, patience and contributions he had for me and which helped me while writing this thesis.

Brno ....

author's signature

CONTENTS

LIST OF FIGURES

LIST OF TABLES

INTRODUCTION

The trends and techniques in information and communications technology are developing and expanding as much as any other fields of human knowledge and possibly even more. Because of unique, recently unthought of demands, new ideas, particularly Internet of Things, Industry 4.0 or SmartGrid have lately been coined and are now well established in the technical speech [1]. Phrases and expressions such as smart home, smart factory, metering automation or advanced energy consumption reduction have all gained in popularity in the recent years and therefore have come to the awareness of each of us. The advance-never-stopping industrial procedures have brought the urge to measure and monitor numerous quantities, based on the field of business, to the foreground. With that, advanced metering systems are being more asked for than ever before.

These infrastructures are designed to communicate with metering devices, for instance gas, electricity, water or heat meters, at a modifiable scheme. By implementing a system of this manner, the customer generally gains many advantages ranging from low maintenance costs and needs to easy meter readout [2]. It may come as a surprise that these systems have already been developed and in many places, they are soon to be deployed or even already running. Furthermore, a considerable part of those devices that are already in duty uses a wireless communication protocol of Wireless M-Bus to communicate with the outer world. What this thesis aims to achieve is a software tool capable of testing these networks via simulating sensor-like behaviour in uni-directional communication with the data collecting servers in test. This tool would be helpful for public utility companies, which could use it for initial testing of concepts, desings and performance of their Wireless M-Bus networks without the need of having any hardware sensors installed.

To accomplish this goal, a solution is presented in five chapters of this thesis. The first chapter introduces terms and expressions, which are important for understanding later parts of the thesis or their purpose. In the second chapter, a broad description of the Wireless M-Bus protocol is provided, focused on the structure of a Wireless M-Bus message, its parts and possible formats, and the architecture of fields it may contain. The next, third chapter, is dedicated to the application created in order to generate Wireless M-Bus data. Fourth chapter describes details of the hardware used in companion with the generator. Lastly, application usage examples and instructions are placed in chapter number five.

1 THEORETICAL INTRODUCTION

In this chapter, uncommon terms used throughout the thesis that a casual reader could find unprecedented are explained.

Notation Throughout the text of this thesis, values of the hexadecimal numeral system will be denoted with `0x` and, similarly, binary numbers will have the prefix of `0b`, i.e., `0x0A` and `0b1010` both deliver the decimal value of 10.

1.1 Industry 4.0

An industrial revolution is a phenomena, in which technological, mechanical or other advances allow for a big leap in current manufacturing standards and improvements in quality, efficiency and production rates that it is a change retrospectively seen big enough for us to call it a revolution. In history, there have been three such major progressions so far:

- the first industrial revolution took place with the transition from hand production methods to machines and making use of steam power in the first half of the 19th century, often called *mechanisation*,
- the second industrial revolution was started by immense utilization of electricity, establishing assembly lines and thus giving birth to mass production at the start of the 20th century, known as *electrification*,
- the third era of industry is connected to the proliferation of computers and the digital age, beginnings of automation and replacement of human workers at the assembly lines, recognized as *digitalisation* [3].

Nowadays, *Industry 4.0* is the fashionable term possibly marking the fourth industrial revolution, the technological leap forward and the paradigm shift which are all expected to come (the name itself being a reminiscence to the conventional software versioning scheme). The most important innovation here is the concept of *smart factories*, which, after the manufactories and human- or machine-operated assembly lines in the preceding eras, introduces a decentralized system of rather independent digital devices, which operate and make decisions on their own while seamlessly communicating and cooperating with each other and with the supervising humans in real time via a wireless – not surprisingly also digital – network. This philosophy is tightly bounded to the idea of *Internet of Things* (see Section 1.2). Building such a factory adds the advantages of interoperability, flexibility, higher degrees of optimization and automation, thus meaning a considerable boost to efficiency and overall productivity, while bringing higher needs for data security, higher-than-present degree of reliability and stability and, on the other hand, unanswered questions of

human jobs losses and capability of integrity maintenance. However, if executed right, Industry 4.0 is expected to bring a boost to economical growth and revenue of up to a third and further reduce operational costs [4].

1.2 Internet of Things

The vision of Internet of Things (IoT) is a world filled with subtle, unobtrusive wireless computing devices serving as sensors, identifiers, messengers or any other technical actors. Such a *thing* in Internet of Things can vary from a broadcasting identifier in a dog leash or vehicle tire pressure sensor to human heart activity monitor or undersea tsunami detector. Generally, it is a device embedded in an everyday object, interconnected with others alike, sending and receiving data based on its purpose. More often than not are these only battery powered, raising high efficiency demands. Building upon recent advancements in microdevice and network technologies (the adoption of IPv6, hardware virtualization, software defined radio...), IoT is an emerging and ubiquitous trend, which, according to estimates [5], about 20 billion devices will belong into by 2020.

1.3 M2M communication

In his book *M2M Communications: A System Approach*, David Boswarthick states: *"Many attempts have been made to propose a single definition of the M(s) of the M2M acronym: Machine-to-Machine, Machine-to-Mobile (or vice versa), Machine-to-Man, etc. Throughout this book, M2M is considered to be "Machine-to-Machine". This being decided, defining the complete "Machine-to-Machine" concept is not a simple task either: the scope of M2M is, by nature, elastic, and the boundaries are not always clearly defined."* [6]. Based upon this definition, Machine-to-machine (M2M) is the communication of stand-alone devices (not peripheral devices like mobile phones, as there is a human in charge) in order to exchange information over a network. M2M communication is also what IoT should be and is using. The key aspects, which distinguish M2M commucination from regular traffic, are M2M's immense **plurality** of endpoints, their **diversity**, decades worth of **longevity** and **battery** dependency.

1.4 M-Bus

Aforementioned smart factories, IoT and M2M, but also home automation or smart metering all need a network protocol to be able to communicate. This protocol

might then be responsible for data collection and distribution, error correction or reporting and overall network management.

Meter Bus (commonly refered to as M-Bus) is a candidate for this. Developed by professor Horst Ziegler in Germany and established by a set of European Norms (found under number EN 13757 and titled *Communication systems for meters*), it has become the European standard for remote reading of meter and service data from water, gas, electricity and many other meters, but it can be applied to security systems and utility control as well. M-Bus is designed as a hierarchical system, consisting of master devices (servers, data collectors), many slaves (*slave* meaning a subordinate device: sensors, data meters) and an interconnecting media (e.g., a two-wire cable). The slaves can be remotely power supplied by this network provided interface [7].

1.5 Cyclic redundancy check

Cyclic redundancy check (CRC) is a method of checking for unexpected errors on transmitted data. The algorithm is not able to correct the errors, it finds out whether they are present or not (with a very high, yet not 100 % certainty). Before sending a message, it is processed with a CRC calculator and the resulting code is appended to the message. Upon receiving the message, the other participating party does the very same calculation, allowing it to compare the codes. If the results match, the message has most likely been transmitted and received error-free. If not, part of the entire message was corrupted with a very slight chance that only the CRC part is broken and the original message itself is not. In case an error is detected, sender is usually notified in order to attempt a resend. CRC verifications are used widely accross the digital communications area [8].

1.6 3 out of 6 encoding

A method of coding arbitrary sequences, standardized along with M-Bus. The procedure divides each byte of data into two halves (4bit nibbles), which are then encoded as 6 bit symbols using a predefined lookup table [9]. This increases information robustness and lowers the error rate while decreasing the communication speed.

1.7 Manchester and NRZ

Both Manchester and Non-Return to Zero (NRZ) are serial encoding mechanisms, more conventional and used than the method of 3 out of 6.

Manchester is employed for its good synchronization recovery abilities and simple processing. As a price for that, a single bit is encoded into a two chip symbol, thus effectively halving the available bandwidth. A logical "1" at the input is represented as "01" at the output and the ingress of "0" as egress "10".

NRZ simply encodes a logical "1" as high value and a logical "0" as low value (or vice versa, alternatively), which provides 1:1 bit-to-chip ratio, but causes synchronization issues with longer sequences of consecutive values [10].

1.8 Binary-coded Decimal

Binary-coded Decimal (BCD) is an information technology technique of encoding a decimal number to binary representation, where a fixed number of bits is used for each digit of the original number, usually four or eight (M-Bus uses 4 bits per decimal digit).

As an example, decimal number 1 289 is with 4-bit BCD encoded as

```
0b(0001 0010 1000 1001)
```

in binary and would be as

```
0b(00001 00010 01000 01001)
```

with an unusual, yet entirely possible and justifiable 5-bit BCD.

2 WIRELESS M-BUS

What this thesis focuses on is not M-Bus itself, it is Wireless M-Bus, which is the radio variant specified in its own file numbered EN 13757-4 (*Wireless meter readout*). Nowadays, it is the more attractive part thanks to the ease of remote readout it allows for.

The network model in use is the same as with wired M-Bus, only the specifications of the physical and link layers are replaced with the newer standard. The application layer on top of these operates in an unchanged way for both [11], which represents an advantage when a transition between wired and wireless networks is needed.

2.1 Motivation

A typical example for this comparison is a regular water meter check performed for a supplying company, which happens regularly in form of a human employee walking door to door, accessing each household's technical room or utility closet, visually reading the desired value with their eyes from the dial and noting it down. This is labour-intensive, error-prone, unreliable (nobody home) and time-consuming [7].

With a technology such as Wireless M-Bus, these readouts do still require a human worker travelling through the country to visit each client, however, this time the data is collected wirelessly, directly into the storing device and without the need of reaching for the meter inside the house. More than that, if the operational range of the meter's transmitter is large enough, the readouts can be performed from a car driving by, for all the customers at the street at once [9].

2.2 Architecture

Since it was designed to be cost-effective, robust and reliable, the M-Bus "network" structure indeed is a bus. Each subordinate device is connected to the shared line and can be therefore plugged or unplugged without limiting the function of the rest. The wireless communication behaves like a broadcast transmission, without any collision avoidance mechanisms. The communication model is based on the ISO/OSI scheme, but, as M-Bus is not a network, the unnecessary layers are not used and are missing from the specification. The same approach was later used when specifying Wireless M-Bus. In Table 2.1, the standard layers' relations to the reality used in wireless or wired M-Bus are shown.

OSI layer equivalent	**Usage**	**EN standard [12] part**
Application	data, values, reports	13757-3
Presentation	not used	not defined
Session		
Transport		
Network	routing and relaying (requires device support), optional	13757-5
Data-link	transfer parameters, frame format, addressing, data integrity	13757-2 (wired), 13757-4 (wireless)
Physical	bit representation, electrical or radio parameters	

Tab. 2.1: OSI layer mapping to M-Bus model.

Complementary to the table, the complete list of standards defining M-Bus (as of 2017) is as follows [13]:

- EN 13757-1 *Data exchange*, which describes the protocol stack, data structures and communication in general.
- EN 13757-2 *Physical and link layer* specifies the first two layers of wired M-Bus, including the addressing, voltage levels and collision detection.
- EN 13757-3 *Dedicated application layer* (DAL), which presents the general application layer, message types, encryption and much more. This DAL is used with wired and wireless M-Bus without much of a difference.
- EN 13757-4 *Wireless meter readout* proposes the basis of Wireless M-Bus: its modes of operation, radio parameters, encoding.
- EN 13757-5 *Relaying*, which portrays options of range extension between devices.
- EN 13757-6 *Local bus* characterizes an M-Bus alternative, which does not belong to the scope of this thesis.
- EN 13757-7 *Transport and security services*, which further defines encryption capabilities, data protection and secure communication.

2.3 Physical layer – working modes

Generally, the lowest, physical layer defines how bits are represented as signals, transmission of these signals and the specifics of this transmission. Wireless M-Bus specifies a number of modes, which then the physical layer specification varies

between. As by the specification, these are Mode S, Mode T, Mode R, Mode N, Mode F and Mode C defining meter-to-contentrator communication parameters, and Mode P and Mode Q with a special purpose, all described below. Some parameters differ while a few stay the same throughout the variants. These were all defined to accommodate different needs when utilizing the protocol between various devices. One important non-static characteristic is the order the bits are sent, which is either most-significant bit (MSB) first, or least-significant bit (LSB) first. MSB is used with modes S, T and R, the rest uses LSB (however, CRC sequences are always transferred with MSB first). They were specified with one in common, though, which is maximizing battery lifetime on the slave side (meters) [14]. A summary of all WM-Bus modes, the usage they were designed for, the encoding and frequency they use as well as the transfer rate they provide is listed in Table 2.2. More detailed descriptions of the modes follow below.

<table>
<tr><th>Mode</th><th>Encoding</th><th>Transfer rate (bps)</th><th>Frequency (MHz)</th><th>Usage</th></tr>
<tr><td>C1</td><td rowspan="5">NRZ</td><td rowspan="2">100 000</td><td rowspan="2">868</td><td rowspan="2">higher data rate equivalent of T</td></tr>
<tr><td>C2</td></tr>
<tr><td>F2</td><td rowspan="2">2 400</td><td rowspan="2">433</td><td rowspan="2">frequent bidirectional transmissions</td></tr>
<tr><td>F2-m</td></tr>
<tr><td>N1</td><td rowspan="2">2 400 or 4 800</td><td rowspan="2">169</td><td rowspan="2">narrow band, long range</td></tr>
<tr><td>N2</td></tr>
<tr><td>P</td><td>Manchester</td><td rowspan="2">not applicable</td><td rowspan="8">868</td><td>router & relay</td></tr>
<tr><td>Q</td><td>NRZ</td><td>time sync</td></tr>
<tr><td>R2</td><td rowspan="4">Manchester</td><td>4 800</td><td>multi-freq readouts</td></tr>
<tr><td>S1</td><td rowspan="3">32 768</td><td rowspan="3">few transmissions per day</td></tr>
<tr><td>S1-m</td></tr>
<tr><td>S2</td></tr>
<tr><td>T1</td><td rowspan="2">3 out of 6</td><td>100 000</td><td rowspan="2">several transmissions per day</td></tr>
<tr><td>T2</td><td>32 768</td></tr>
</table>

Tab. 2.2: Summary of Wireless M-Bus modes.

a) Mode S

Represented by submode S1, which was designed for stationary usage. Unidirectional communication is possible between the meter and a stationary positioned receiver. In this mode, the meter sends data without checking whether there is a device to receive them – there is no answering. Manchester encoding and long header

(explained later) are used. This mode is suitable for only a few transmissions per day. Associated submode S2 represents a bidirectional variant of the S1 mode with only a minority of parameters altered – Manchester encoding with short header are used. In addition, there is mode S1-m with short header only, thus requiring a continuously ready receiver [15]. A simple topology schema, showing possible wireless connections of Mode S, is portrayed in Figure 2.1.

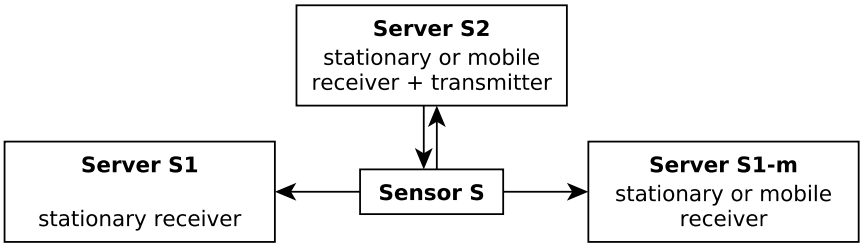


Fig. 2.1: Schema of Wireless M-Bus Mode S, submodes S1, S1-m, S2 operation.

b) Mode T

While in the frequent transmit mode, T1, the meter transmits a short telegram[1] regularly and frequently. This behaviour enables for data readout while only walking or driving by. This is a unidirectional communication. The intervals between activity can range from only several seconds to a few minutes. 3 out of 6 encoding is used along with short header. T1's derivative, Mode T2 brings bidirectional communication option. After transmitting, the meter waits for a short period of time for an acknowledgement telegram. Upon receiving one, bidirectional communication can proceed. 3 out of 6 encoding and short header are used as well [16]. Transmission options provided by Mode T are presented in Figure 2.2.

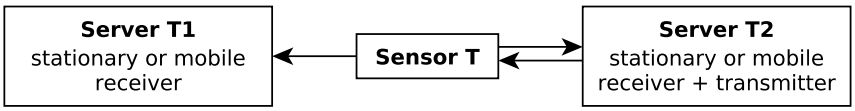


Fig. 2.2: Schema of Wireless M-Bus Mode T, submodes T1, T2 operation.

[1]Messages with content from at least data-link or application layers are referred to as *telegrams* in M-Bus.

c) Mode R

Frequent receive mode R2 defines that meters be listening for a wake-up message (usually every few seconds), which then triggers further cooperation. This mode does not suppose spontaneous sending of data. Since there is no data exchange unless bidirectional connection is established, there is no unidirectional R1 mode. Manchester encoding and short size header are used [16]. A diagram of connection in Mode R is portrayed in Figure 2.3.

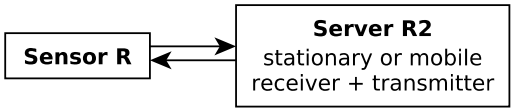


Fig. 2.3: Schema of Wireless M-Bus Mode R operation.

d) Mode N

This introduces an option optimized for narrowband operation with an extended range. It is made for long-distance uni- or bi-directional communication with stationary receiver. Non-return to zero encoding is required. Again, there are N1 and N2 descendants [15], both displayed in Figure 2.4.

Fig. 2.4: Schema of Wireless M-Bus Mode N operation.

e) Mode F

Mode F is always bidirectional and stands for both frequent transmit and receive. Two submodes exist, differing by the side that initiates the communication. If this role is on the meter (F2-m), this mode resembles T2, if on the data collector (which uses the wake-up frame to achieve this), then Mode F2 resembles Mode R2. It is also suitable for long range communication [16]. Both communication types available in Mode F, distinguished by the order of initialization messages, are portrayed in Figure 2.5.

Fig. 2.5: Schema of Wireless M-Bus Mode F, submodes F2, F2-m operation.

f) Mode C

C (compact) mode uses more efficient encoding techniques to achieve higher data rates with energy consumption similar to that of Mode T. Most of the attributes were inherited from this mode (only NRZ encoding is used instead of Manchester). Therefore, C1 and C2 represent modes similar to T1 and T2 [16] respectively and provide higher throughput.

g) Mode P

P is a special mode, as it does not specify any communication between meters and a concentrator. A device in P mode has a role similar to that of a router. It changes addresses of transferring telegrams and routes them to their destination in a larger M-Bus network [14]. Topology of a simple network with Mode P routers is outlined in Figure 2.6.

h) Mode Q

Similarly to Mode P, Q mode defines a whole other function of a concentrator. If in Q mode, device acts as a time information distributor throughout the network with latency, sleep durations and other aspects taken into account [14]. The operation of server, working in Mode Q, is portrayed in Figure 2.6.

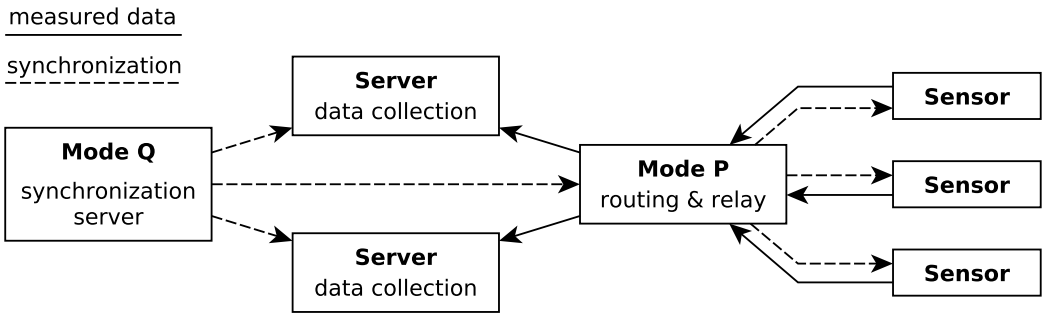


Fig. 2.6: Schema of Wireless M-Bus Modes P and Q operation.

2.4 Data-link layer

The second lowest layer is the first to introduce a network model abstraction. It provides services for the higher layers, while utilizing the services of the physical layer. Thus, it transforms application data to the medium. Other responsibilities include generation of the CRC code for outgoing telegrams, its verification for incoming telegrams, correction in case of errors (when possible), structuring the frames from data given by the application layer and tracking the acknowledgement frames. There are several information fields defined in the standard, which can be classified as Data-link layer payload. These are described in the following section.

2.4.1 Information fields

Fields are sections of the telegram, designated to contain specific parts of the whole telegram information. These fields have then an exactly given spot and structure within the unit. (Note: the fields below are **not** listed in the order of transmission.)

a) Preamble

A known sequence of bits transmitted at the beginning of every message with the purpose of synchronization. The form and length differ from mode to mode and may depend on the communication direction, too. It ranges from 3 to 72 bytes in size: [14]

- 4 bytes for Mode N (non-4GFSK modulated),
- 6 bytes for Modes T and S (short),
- 8 bytes for Mode C and Mode N (4GFSK modulated),
- 12 bytes for Modes F and R,
- 72 bytes for Mode S (long).

As an example, the preamble for mode R2 (independent on the direction) would look like this:

```
0b([01] 0100 0111 0110 1001 0110)
```

with the two bits in brackets repeated 38 times, making for the total length of 12 bytes.

b) A-field

Carries the device address, which is 6 bytes in length and unique for every device. If this field is contained within a *send* or *request* frame, the address belongs to

the meter or receiver that sends the frame. If, on the other hand, it is a *confirm* or *response* type of telegram, the address points to the device that originated the exchange, which is the receiver, in these cases [17]. Information pieces contained in this field are device serial number, device version and device type (warm meter, gas meter...).

c) C-field

Determines what type of action or service is being demanded by sending this frame. The options for this field are, amongst others, *send*, *confirm*, *request* and *response*, each with own further division [17].

d) CI-field

CI – control information – holds the specifics of the application data type in the application payload. It provides information for the superfluous layers, such as COSEM [14]. Using the options in this field, the server can, for instance, change the communication baud rate or force synchronization, and the sensor is able to report alarms and errors.

e) CRC

Contains the 2-byte long verification checksum. In case of Wireless M-Bus, this is always calculated using the following polynomial: $x^{16} + x^{13} + x^{12} + x^{11} + x^{10} + x^8 + x^6 + x^5 + x^2 + x^0$ (also know as CRC-DNP polynomial [18] from Distributed Network Protocol). The calculation starts with zero and deals with the MSB first. The resulting 16 bits are also transferred with MSB first [17].

f) Data field

Section designated to hold application layer payload of unspecified length. It has its own categorisation and numbering described later in Section 2.5 [14].

g) L-field

Indicates the length of the frame, excluding the length of the L-field itself and that of the CRC field (thus this effectively represents the size of the link layer payload). This information is calculated by the data-link layer upon transmission and used by the counterpart at reception [14].

h) M-field

Field designated to manufacturer's identification. Each manufacturer with the intention of providing M-Bus-enabled hardware is assigned a three-letter ASCII code (the list can be found at the DLMS User Association website [19]). Using this code, the binary representation is calculated by an algorithm specified in the M-Bus standard [12]. In this procedure, the three letters from the code are processed consecutively from left to right and the resulting hexadecimal (or binary) output is given by the formula below:

$$\begin{aligned} MAN = {} & [\alpha(\text{1st letter}) - 64] \cdot 32 \cdot 32 + \\ & [\alpha(\text{2nd letter}) - 64] \cdot 32 + \\ & [\alpha(\text{3rd letter}) - 64], \end{aligned}$$

where *MAN* is the final 2-byte code (which can later be used in the telegram) and the function α(x) transforms letter x into its ASCII numeric code [17].

i) Postamble

Postamble (also trailer), similarly to preamble, is a sequence of bits specified in advance and unchanged later, used to announce the end of the telegram. Only some modes use this field. If it is present, though, it is a pair of alternating bits repeated several times [14].

2.4.2 Frame formats

Two frame formats are specified in the respective EN-13757-4 document – Format A and Format B. Their structure is similar, both are described below.

a) Format A

Format A is general and can be used in all modes, which were explained earlier. The structure [16] of an A frame is shown in Table 2.3. It includes a preamble, whose length is mode-dependent (see Section 2.4.1 a)) and which is required.

Frame A				
Preamble	Block I	Block II	additional blocks (optional)	Postamble

Tab. 2.3: Wireless M-Bus: structure of Frame Format A.

After the preamble, Block I follows, with contents as seen in Table 2.4. These are all the information fields with given length, finished with the CRC computed from them (C-, M- and A- fields are treated only).

Block I				
L-field	**C-Field**	**M-Field**	**A-field**	**CRC**
1 byte	1 byte	2 bytes	6 bytes	2 bytes

Tab. 2.4: Wireless M-Bus: structure of Frame Format A, block I.

Block II (structured as in Table 2.5) is the first to carry *useful* data, marked with control information field and hashed in a CRC.

Block II		
CI-field	**Data field**	**CRC**
1 byte	15 bytes maximum	2 bytes

Tab. 2.5: Wireless M-Bus: structure of Frame Format A, block II.

Maximum length of a single data field is 15 bytes. If more than that needs to be transmitted in a single message (for administrative or power-saving purposes), an arbitrary number of additional data blocks can follow (Table 2.6), unless the total maximum frame length is exceeded. Frame of Format A is ended with a postamble.

Optional blocks	
Data field	**CRC**
16 bytes maximum for each	2 bytes

Tab. 2.6: Wireless M-Bus: structure of Frame Format A, aditional blocks.

b) Format B

Frames of Format B [9] are an option only for modes C and F. This format was designed in order to reduce the unnecessary overhead and thus increase battery life or space for goodput. As its origins are in Format A, these two are more similar than not.

Frame B			
Preamble	Block I	Block II	additional block (one or none)

Tab. 2.7: Wireless M-Bus: structure of Frame Format B.

Frame of Format B starts with the same preamble as frame of Format A also followed by Block I, which is – compared to frame Format A-like block I – missing the final CRC field.

Block I			
L-field	**C-Field**	**M-Field**	**A-field**
1 byte	1 byte	2 bytes	6 bytess

Tab. 2.8: Wireless M-Bus: structure of Frame Format B, block I.

Block II also has the same form, but the length of data field, which is allowed to be sent without being followed by a CRC, is significantly increased from 15 to 115 bytes (increased goodput to overhead ratio).

Block II		
CI-field	**Data field**	**CRC**
1 byte	115 bytes maximum	2 bytes

Tab. 2.9: Wireless M-Bus: structure of Frame Format B, block II.

The tailing Block III is optional and meant to be used if the desired data to be sent are longer than the allowed maximum in Block II. After that, there is no postamble or additional data blocks.

Optional block	
Data field	**CRC**
126 bytes maximum	2 bytes

Tab. 2.10: Wireless M-Bus: structure of Frame Format B, optional block.

2.4.3 Example

Suppose a telegram of frame Format B, written in hexadecimal and stripped from its preamble and postamble (these are known in advance and too long for this illustration), which looks as follows:

```
17 44 01 6A 8A 39 02 00
01 06 72 8A 39 02 00 01
6A 01 06 AA 00 01 01 DA.
```

Bellow, there is the list of information that the receiving device can gain about the telegram and the data it holds.

- `0x17` is the L-field, holding the total length of the rest of the link-layer payload (C, M, A and CI fields and data). That is 23 bytes in this case.
- `0x44` is the C-field, indicating the function of the telegram: "SND-NR", *send/no reply*.
- `0x(01 6A)` is the M-field. Since it has more than one nibble, the endianness begins to matter. This field should be ordered with LSB first, so the actual value of this block is `0x(6A 01)`, which delivers the manufacturer ID of "ZPA".
- `0x(8A 39 02 00 01 06)` are 6 bytes of the A-field. The first four of them signal the identification (`0x(00 02 39 8A)`), the next one the version (version `0x01`), and the last one is for the type of the meter (`0x06` means warm meter).
- `0x72` is the CI-field, saying that data in this telegram have full header (12 bytes in length).
- `0x(8A 39 02 00 01 6A 01 06 AA 00 01 01)` represents the 12 bytes of the full header. This already belongs to the application layer. Here, the identification number, manufacturer code, version and device type are repeated, followed with the access number (`0xAA`, serves as counter), status (`0x00` – no errors) and signature (`0x(01 01)` – this has meaning for encryption and data privacy), also called *configuration field*.
- After these fields, the actual data would follow (substituted with `0xDA` in this example).

2.5 Application layer

For the application layer, CI-field is very important. Prepending the rest of the payload, it indicates the formatting of the rest of the data and their purpose (*Synchronize action* and *Slave to master: report of alarms* are examples of this). What has so far been denoted simply as data field will now be broken into parts and explained. Firstly, there is space designated for data header, which is followed by a data record header and data themselves, as portrayed in Table 2.11.

Application layer data		
Data header	**Data record(s)...**	
	Data record header	**Data**
0, 4, 12 byte variants	unspecified length	unspecified length

Tab. 2.11: Wireless M-Bus: structure of application layer data.

Each telegram has a data header. The two fields after it form a couple and there can be more of these couples – each with data record header and data, all transferred within one frame, as numerous as the total length maximum allows for. The limit of a single telegram size is set to 255 bytes to ensure as much interoperability and compatibility with other data-link and application layers as possible.

2.5.1 Data header

The application layer recognizes three types of data headers: none, short and long. Data headers precede data records in the telegram structure.

a) Omitted data header

If the CI-field signals hexadecimal `0x78`, there is no data header available in the telegram. Based on that, no encryption is possible.

b) Short data header

Signalled with hexadecimal `0x7A`, short data header takes 4 bytes of space and it is usually used once a communication has been established and additional information would be redundant. It consists of three fields:

- Access number (1 byte), holds an incrementing number, which supports lost frame detection and repeated frame inclusion. Its starting value is chosen randomly when communication begins.
- Status code (1 byte), whose purpose depends on the direction of the telegram it is in. If the telegram was sent by a meter, this field indicates alarm and error states. If the direction is opposite, it bears information about Received signal strength indicator (RSSI) – basically indicates quality of reception – of the last frame.
- Signature (2 bytes), also known as configuration field, which contains information about desired encryption mode, specifics of this encryption and length of the encrypted data.

Short data header		
Access number	**Status code**	**Signature**
1 byte	1 byte	2 bytes

Tab. 2.12: Wireless M-Bus: structure of DH, short.

c) Long data header

Long header takes over the three fields from the short header and adds four more, which have already been described as well, in the Section 2.4.1. In the order of transmission, they are:

- identification number (4 bytes), which is also the first part of the A-field,
- manufacturer ID (2 bytes), reflects the M-field,
- version (1 byte), also fifth byte in the A-field,
- meter type (1 byte), also sixth byte in the A-field,
- access number (1 byte),
- status code (1 byte),
- configuration field (2 bytes), which have been described in the short header paragraph above.

Long data header						
Serial number	**Man. ID**	**Version**	**Meter type**	**Access number**	**Status code**	**Signature**
4 bytes	2 bytes	1 byte	1 byte	1 byte	1 byte	2 bytes

Tab. 2.13: Wireless M-Bus: structure of DH, long.

The CI-field code for long header is `0x72` and it is 12 bytes in length.

2.5.2 Data record header

Data record headers (DRHs) hold meta-information about the data. One DRH is further divided into Data information block (DIB) and Value information Block (VIB), as seen in Table 2.14.

Data record header			
Data Information Block		**Value Information Block**	
DIF	DIFE ...	VIF	VIFE ...
1 byte	1 byte each (up to 10 times)	1 byte	1 byte each (up to 10 times)

Tab. 2.14: Wireless M-Bus: structure of DRH.

a) Data Information Block

Each DIB contains exactly one Data information field (DIF) and can be optionally extended by a maximum of ten Data information field extensions (DIFEs).

DIF In a DIF, 8 bits of information are stored, listed from the most significant bit to the least one:

- extension bit, which indicates, whether there is or is not a DIFE after this DIF,
- storage number bit, which transfers the LSB of the storage number,
- function field (2 bits), which expresses the type of value being transmitted (options here are *instantaneous*, *minimum*, *maximum* and *error value*),
- data field (4 bits, not to be mistaken by name with the data-field in frame from Section 2.4.1), carries information about the data length and coding.

Data information field			
Extension bit	**Storage num. LSB**	**Function field**	**Data field**
1 bit	1 bit	2 bits	4 bits

Tab. 2.15: Wireless M-Bus: structure of DIF.

DIFE A DIFE also contains an extension bit, marking whether there is another DIFE after this one, and it stores additional information about tariff and subunit of the device, as well as more bits of the storage number.

Data information field extension			
Extension bit	**Device subunit**	**Tariff**	**More storage number bits**
1 bit	1 bit	2 bits	4 bits

Tab. 2.16: Wireless M-Bus: structure of DIFE.

b) Value Information Block

The structure of VIB is very similar to that of DIB's. Within a DRH, VIB always takes place after the DIF and all possible DIFEs. It also consists of precisely one VIF and none up to ten VIFEs.

VIF VIF's primary purpose is to deliver information about the metered unit (e.g., kilowatt hours) and its multiplier (e.g., 10^6 in case of mega kilowatt hours). Thus, it only consists of:

- one extension bit,
- seven bits of unit and multiplier information.

Value information field	
Extension bit	**Unit and multiplier code**
1 bit	7 bits

Tab. 2.17: Wireless M-Bus: structure of VIF.

VIFE In VIFE, given that there is one extension bit as well, the whole 7 bits left are all taken by one of special codes, defined in a table as part of the standard [20].

Value information field extension	
Extension bit	**Special purpose code**
1 bit	7 bits

Tab. 2.18: Wireless M-Bus: structure of VIFE.

2.5.3 Data

Introduced via the data record header, raw data follow. Albeit their structure is outlined within the standard as a recommendation, many of the manufacturers implement proprietary solutions for this. There are, however, many homogenized value structures, which can be used to encode many of the common data sets. These are called *Type*s and the original WM-Bus standard EN 13757-3 [20] of 2004 defines 12 of them, labeled alphabetically as Type A through Type L. List of the types and their purpose is outlined in Table 2.19. Types F and G are amongst the most utilized and are implemented in the generator as well. Therefore, Type F is described in more depth below. Type G, which only delivers a subset of the information found in Type F (date and time compared to only time), is structurarily very similar.

Type F The construction provided by Type F is used for storing date and time information with the precision of minutes (Type I further narrows this down to seconds). Separate parts of the entire information are placed in the final 4 byte structure as per Table 2.20, where roman numerals in the table substitute for:

- (i): bit reserved for future use,
- (ii): bit signifying values validity (0 for valid),
- (iii): bit for standard / summer time (0 for standard).

Since there is only 60 different possible values for the minute number and at the same time, there has to be a space of whole-number of bytes, able to accomodate at least 60 bits to store them (64), there are unused amounts (60, 61, 62 and 63 in this case). These can be used to store special cases, such as *every minute repetition* with

Type	Stored value	Length [bits]
A	Binary coded unsigned integer	8
B	Binary coded integer	8
C	Unsigned integer	8
D	Boolean value	1
E	Deprecated, unused	
F	Date and time (minute, hour, day, month, year)	32
G	Date (day, month, year)	16
H	Floating point number	32
I	Date and time (second, minute, hour, day, month, year)	48
J	Time of day (second, minute, hour)	24
K	Instructions regarding daylight saving	32
L	Precise mode scheduling (sleep, listening)	88

Tab. 2.19: Wireless M-Bus: defined data types.

the value of 63, which could not be achieved otherwise. Similar concept is applied for hours, days, months and years.

As an example, the date of 4. July 2014 8:03 is processed as follows:

- minute (3) = `0b000011`,
- hour (8) = `0b01000`,
- day (4) = `0b00100`,
- month, counted from zero (6) = `0b0110`,
- year (14) = `0b0001110`,
- centuries after the nineteenth (1) = `0b01`.

and finaly coded into

`0b(00010110 11000100 10101000 10000011)`,

where value bounderies are marked by underlined / overlined sections for easier distinguishment.

2.6 Structure summary

The structuring of Wireless M-Bus telegrams, as it has just been explained, can be a bit difficult to comprehend with its multi-tier structure and naming. To cover that, a brief summary is given below.

Byte 1	**Bit**	8	7	6	5	4	3	2	1
	Data	(ii)	(i)	minute					
Byte 2	**Bit**	16	15	14	13	12	11	10	9
	Data	(iii)	century		hour				
Byte 3	**Bit**	24	23	22	21	20	19	18	17
	Data	year (1/2)			day				
Byte 4	**Bit**	32	31	30	29	28	27	26	25
	Data	year (2/2)				month			

Tab. 2.20: Wireless M-Bus: Type F in detail.

Two frame formats exist in Wireless M-Bus: more standard Format A and a shortened Format B. In frame of Format A, there are (in order): preamble, L-field, C-field, M-field, A-field and their cumulative CRC, CI-field, data and their CRC, optionally more data with CRC, and a postamble. Compared to that, frame of Format B misses on the first CRC and the postamble, thus having more space for data.

The data part consists of Data header (DH), which is not mandatory and distinguishes short and long forms, Data record header (DRH) and data. DRH and data together form a Data Record (DR). The DRH has two parts, Data information block (DIB) and Value information block (VIB). In a DIB, there is always one Data information field (DIF) and up to ten Data information field extensions (DIFEs), similarly in a VIB, there is always just one Value information field (VIF) and also up to ten Value information field extensions (VIFEs). The data themselves (payload) follow. There can be more DRs in one telegram.

2.7 Encryption

Encryption is supported in WM-Bus, however, it only affects part of the application layer data. The information about the encryption that has been used to encrypt the telegram is stored in the signature field (2 bytes), which is the last word of the data header (both short and long one). Data *preceding* this field are always unencrypted, as well as the signature field itself. All data *following* the field may be partially or fully encrypted, or they may not (in case no encryption was used). The length of the encrypted data is stored in the lower byte of the signature word, while the higher byte contains a code indicating the method used for encryption. In case no encryption was used, both parts of the word contain a zero (`0x00`). When only a part of the payload is to be encrypted, it comes right after the signature word,

followed by the unsecured part. This allows for encrypting only the private part of the message and thus further saving power (e.g., timestamps might not need to be protected).

Encryption techniques of Advanced Encryption Standard (AES) and Data Encryption Standard (DES)[2] are supported. M-Bus mandates the use of Cipher Block Chaining (CBC), which is available in both of the standards. Recognized encryption modes are as following:

- mode 0: no encyption,
- mode 2: DES with CBC, zero initialization vector (IV),
- mode 3: DES with CBC, non-zero IV,
- mode 4: AES with CBC, zero IV,
- mode 5: AES with CBC, non-zero IV,
- modes 1, 6, 7: reserved for future use [21].

2.8 Real data structure example

To verify the outlined structure of Wireless M-Bus telegrams, a real-traffic capture [22], containing a message from a WM-Bus warm meter, was parsed and analyzed. Extracted message had been sent from a device manufactured by Bonega. From the analysis results listed in Table 2.21 it is clear, that the manufacturer obeys the standard data structure only partly. Fields L, C, M, A and CI all comply to the WM-Bus definition by their order and contents, the CRC Fields are placed correctly as well. So is the data header and access number, status byte and signature field in it, however, the two data records are not. The data field conveys the information of temperature measurement together with exact minute this action was taken, but Value information fields (VIFs) are prepended to the data sections, which is not as per the standard. The telegram in question had the full content as follows:

```
0x(20 44 EE 09 21 01 00 00 01 06 FC E2 7A 4F 00 10
05 1A B9 4C 4F Da 69 43 09 E3 47 E8 6F A4 37 79 0C)
```

and had been encrypted using AES and cipher block chaining (CBC) with the encryption key of

```
0x(2B 7E 15 16 28 AE D2 A6 AB F7 15 88 09 CF 4F 3C)
```

[2] As later versions of the M-Bus norm acknowledge, DES is an obsolete and surpassed standard and should not be used [16].

and initialization vector composed of specific parts of the telegram and a padding with the value of

```
0x(EE 09 21 01 00 00 01 06 4F 4F 4F 4F 4F 4F 4F 4F).
```

Note that from the whole message, only a part of the application layer payload is encrypted in Wireless M-Bus. In case of this telegram, the encrypted section starts with the byte of `0x1A` and spans to the end (last 16 bytes).

2.9 The role of Wireless M-Bus in Industry 4.0

M-Bus (wired) was developed in the early 1990s and extended with its wireless form in 2005 (when the first draft of the EN 13757-4 was published, approved a year later [12]), which is 5 years before the terms of IoT and Industry 4.0 started gaining on popularity in 2011 and even longer before they got to attention of the mass market in 2014 [23]. Despite that, Wireless M-Bus presents a solid competition to protocols and networks tailored just for Industrial Internet of Things (IIoT, which is a specific subset within regular IoT and represents more or less a synonym for Industry 4.0), such as SigFox, LoRa or NB-IoT. Since it follows very similar goals (sensor-independence, battery-longevity, meter-automation), but holds a few years advantage, it is even better established, settled and stabilised than most of its competitors [24].

For the reasons stated above, no drop in WM-Bus popularity is expected within the Industry 4.0 era, moreover an **important role** is assumed for it, however constrained to metering applications it is.

Offset [B]	Field	Hex value [decrypted]	Meaning
0	L	20	There are 32 bytes after this one
1	C	44	C code is "SND/NR" (Send/No reply)
2	M	EE	Manufacturer's ID is 0x(09 EE),
3		09	decoded as BON, short for Bonega
4	A	21	Device ID (serial number) is 0x(00 00 01 21)
5		01	
6		00	
7		00	
8		01	Device version is 0x01
9		06	Device type is 0x06 (warm meter)
10	CRC	FC	Checksum for fields L–A
11		E2	
12	CI	7A	CI code is "Short header" (4 bytes)
13	DH	4F	Access number (sequential)
14		00	Status byte (no errors)
15		10	Signature: encrypted data length is 0x10,
16		05	(16 B), encryption type is 0x05 (AES with IV)
17		1A [2F]	AES filling
18		B9 [2F]	
19	VIF I	4C [04]	Value type: Type F (Date and time)
20	VIF II	4F [13]	Value type: Flow temperature (10^{-2} °C)
21	Data I	DA [1A]	Temperature value is 0x(22 1A),
22		69 [22]	this is 8 730 in decimal and with
23		43 [00]	the multiplier (10^{-2}) applied,
24		09 [00]	the reading was 87.3 °C
25	CRC	E3 [04]	Checksum for fields CI–Data I
26		47 [6D]	
27	Data II	E8 [03]	Data encoded in Type F structure
28		6F [28]	(see Section 2.5.3 for details),
29		A4 [C4]	holding the time definition of
30		37 [16]	6 April 2014, 08:03
31	CRC	79 [2F]	Checksum for field Data II
32		0C [2E]	

Tab. 2.21: Bonega telegram analysis.

3 GENERATOR

As the practical part of this thesis, an application with the purpose of generating Wireless M-Bus data was created. It runs on Java 8 and uses JavaFX 8 to provide graphical interface. Since Java is designed to be a multiplatform language, this application can theoretically run everywhere where a Java virtual machine (JVM) runs. The real aim in this case are Windows, Linux and Mac operating systems, though, thus covering the vast majority of desktop workstations.

3.1 Application

The generator adapts the WM-Bus fields hierarchy and, partly, uses a similar structure to achieve as close and simplistic implementation as possible. In terms of that, it operates with single fields and their parts when constructing the data (embodied via Java objects), and with whole frames or telegrams when manipulating or sending them. Full UML-like class diagram is attached in the Figure A.1 in appendix A. Brief explanation on the key parts is provided below.

The basis of the program's operation can be described followingly. Classes, which manipulate the Graphical User Interface (GUI) or one of the Command-line User Interfaces (CLIs) only have the purpose of allowing data input and controlling the transmission actions. The core logic is located elsewhere. Thanks to this, boundaries for the interface-specific parts are small and more generic implementation can be achieved. When a telegram is constructed using one of the user interfaces, it is stored in memory as an instance of class derived from *Frame*, with all the details accessible and manageable. These are attached in the form of objects implementing the interface of *Field*. Structure enforcement, value transformation and validation are delegated to specific *Field* instances. When a request for connection (to the target communication output channel) is specified and raised, an object of class inherited from *Sender* is created. This instance is responsible for everything regarding the periodical sending process, including the transmission scheduling and initializing, controlling and retaining the channel. By extending presented base structures of *Frame*, *Field*, *Sender* and others, the implementation and functionality can be expanded to meet new requirements

3.1.1 *Frame* base class

Frame is an abstract class, designed to provide similarities between both kinds of frames (so far in WM-Bus these are Frame A and Frame B – both described in Section 2.4.2). An instance of *Frame* holds a collection of *Field*s stored within

the frame and dictates several methods that its subclasses have to implement. It also provides routines, which can be applied to both frames without a difference. Amongst them, there are:

- *construct()* method, which collects all known parameters for the frame, places them in the correct order and returns a list of fields within the frame corresponding to that,
- *getBytes()* method, which builds the bytes of the entire constructed frame and returns them in a list,
- format-independent generic getters (getter is a method for obtaining object properties without actually accessing the implementation),
- format-independent generic setters (methods for changing object properties without the access to other values and inner implementation).

Classes extending *Frame* (*FrameA* and *FrameB*) are responsible for constructing frames based on demands and parameters (addressing, payload, length ...) and delivering their byte representations, but most importatnly, they hold the *Field*s a provide them upon request.

3.1.2 *Field* interface

Field is a very simple interface. It serves a similar purpose as *Frame*, one imaginary step deeper (a *Frame* holds *Field*s, not vice versa). There is not many similarities between diferrent fields, which could be brought together in this interface, which is why it only is an interface and not a class. However, it presents a so called *marker interface*, enabling all fields to be treated as equal.

Classes, which implement *Field* (*AField*, *MField*, *DataField*, *DataRecord*, *Preamble* and others) are then used to accept respective defining parameters (manufacturer ID in case of *MField*, bytes to be encompassed in case of *CRCField*) and, while obeying the WM-Bus standard, construct the bytes representing the parameters. *CIField* can serve as a simple example: its constructor expects one of the CI-field codes and transforms it into hexadecimal form, exactly as it is found in the telegram. Full class diagram of all fields implementing *Field* can be found in Figure A.2 in the appendix A.

3.1.3 *DataField* class

Although Data field is just one of the fields M-Bus recognizes, the class *DataField* is by far the most important *Field* implementation in this program. It holds a reference to one *DataHeader* object and a list of references to *DataRecord* objects. Furthermore, *DataRecord* holds a *DIF*, a *VIF* and multiple *DIFE*s and *VIFE*s (just as they are found in WM-Bus), which eases the manipulation. Because these have

a number of specifics in common (compared to other classes), a base class was created for them. It is called *DataRecordHeaderBase* and only the classes *DIF*, *VIF*, *DIFE* and *VIFE* extend it. A class diagram focused on *DataField*, its attributes and their hierarchy is shown in Figure 3.1.

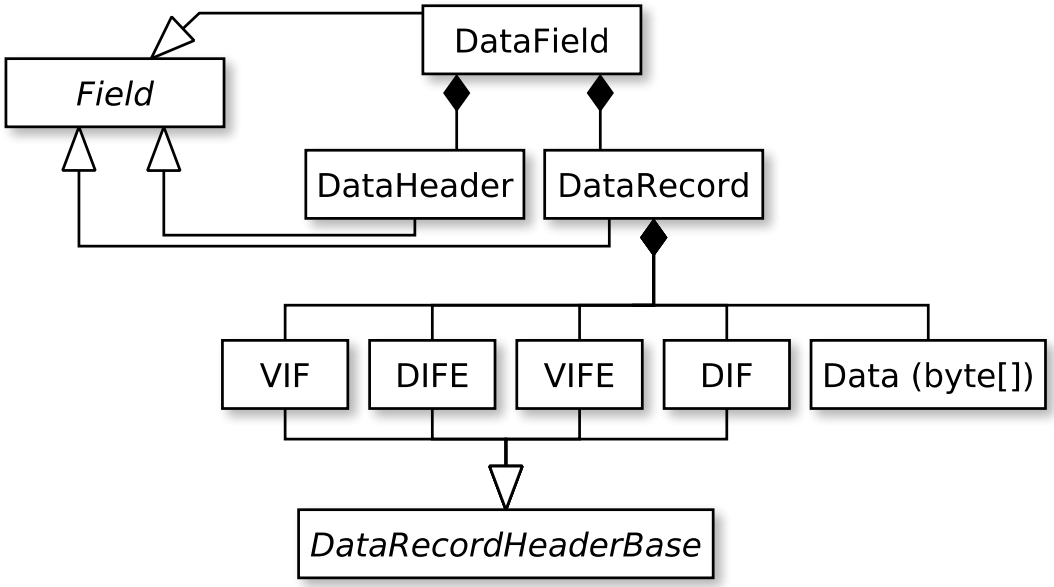


Fig. 3.1: Selected relationships of class *DataField*.

This class also handles encryption – note that only application layer payload is encrpyted in Wireless M-Bus. For this reason, there is an extra class called *EncryptedDataRecord* extending *DataRecord*, which is created when encryption is used and only stores an array of encrypted bytes. An object of this class can then be placed and used anywhere where an object of class *DataRecord* is expected and thus, in the higher layers of the program, no distinguishing has to be made (although the option is still present thanks to Java's Run Time Type Identification (RTTI) mechanism). In the holding class (*DataField*), there is a method called *encrypt*, which reads bytes from the unencrypted *DataRecord* field object and substitutes it with the encrypted one.

3.1.4 *Sender* base class

Sender is another abstract class. Its task is to manage periodic sending of telegrams, for which it features methods such as *schedule* (registers given telegram for sending),

reschedule (changes the sending period of an already registered telegram) or *shutdown* (terminates all periodic operations, closes opened connections and returns). It does not specify, how or where should the telegrams be sent, it only manages the registering and periodicity. The task of transmission execution is delegated to inherited classes. So far, there are two classes extending *Sender*: *UDPSender*, which adds a simple implementation of sending the telegram bytes through IP network from and to a specified UDP socket, and *CommSender*, which operates similarly on serial ports. Since communicating with a device via a serial port is not a straightforward, nor simple procedure, there is an extra class of *IqrfDeviceModel*, which manages this task for the rest of the program. Details of this class are described in Section 3.1.6.

3.1.5 *PeriodicTelegramBase* base class

Instances of subclasses of abstract *PeriodicTelegramBase* are auxiliary models, which are viewed from the GUI in place of "raw" *Frame*s and are used by *Sender* objects for the transmission. There are two classes inheriting from *PeriodicTelegramBase*, one for GUI (*GUIPeriodicTelegram*) and the other one for both CLIs (*CLIPeriodicTelegram*). They represent a role similar to *Frame* instance, extended with the information about its transmission period, transmission status, whether it should be fluctuated before sending and whether it should be sent periodically at all. There is also a unique identifier, used by the scheduling tools, and other interface-specific details and logic. When a *Frame* is defined by the user, it is, at some point, encapsulated into a *PeriodicTelegramBase* object, which further specifies its transmission-related properties. This relieves the *Frame* class from doing so, thus it can focus on its designated task (which is managing collections of *Field*s).

3.1.6 *IqrfDeviceModel* class

This class creates an abstraction level above a specific hardware device – IQRF module TR-72D-WMB. It has all the information needed to control the module, read and change its configuration, send service commands and other instructions to the chip. It is also responsible for establishing a communication link with the device – if it is connected – and mediating all communication from the program to the device and vice versa. For the serial port connection, transmission and reception of data the model uses the open-source library of jSSC, described in more detail in Section 3.2.2.

It provides many methods for setting specific parameters of the module, reading it, sending well-defined commands or unstructured bytes, restarting it, putting to sleep and so on. The inputs for this class are actions which other parts of the

program would like to perform on the device, and its outputs are performing these actions within the device. Because of this, it behaves more like a library, delivering the knowledge of controlling this specific hardware.

As various requests for reading of module configuration come from the application or needs to do so arise for different reasons, an instance of *IqrfDeviceModel* keeps the information it has gained about the state of the device and uses them as a simple cache. Thus, when a reading request is made to a parameter that the class knows has not changed since its last reading (because it would have been the entity to change it), it simply does not perform the reading again, but uses the cached value. This behavior saves nonnegligible amounts of time, as retrieving the information from the module (from requesting it to actually obtaining it) is never instant (various measurements have been performed and the results are presented in Section 4.8).

3.1.7 *ByteUtils* class

ByteUtils is a utility class made to accommodate static methods which operate various byte or bit related structures. These methods are static – do not depend on an instance of the class and only operate with parameters, and public – can be called from anywhere in the application (in fact, their accessibility goes beyond the application). Amongst the tool methods included in this class are:

- `String bytesToHex(Object bytes)`, which receives an array of primitive bytes, an array of non-primitive Byte objects or a List of Byte objects and converts their values into hexadecimal text (e.g., for showing device serial number to the user),
- `Byte[ ] hexToBytes(String s)`, which is the opposite of the previous one; a version with desired length specification is included as well, so it pads the output with zeros if the input is shorter than required (i.e. for accepting hexadecimal input from the user),
- `String byteToBits(Byte b)` to convert a single Byte object into its binary representation (for simpler binary arithmetics),
- `String intToBitsWithFixedLength(int i, int length)`, which converts given integer number into binary text and ensures it has given length (used e.g., for coding Type F (see Section 2.5.3)).

3.1.8 Enumerations

A number of enumerations is declared in the scope: status codes (values such as *Application busy* or *Power low*), data field codes (*No data*, *8 digit BCD*) and value types (*Volume*, *External temperature*) are only examples of that. They are mappings

of standardized Wireless M-Bus tables to programmatically used entities. Besides the value name, more information is usually stored in an enumeration. In most cases found in this application, it is at least the hexadecimal value for the code or its bit representation, as per WM-Bus standard, and a so-called nice name, which is a user-friendly version of the value description. An example can be picked from the *DataFieldCode* enumeration, one of whose rows looks like this:

```
Integer24("0011", 24, "24 bit integer / binary"),
```

where the following applies:

- *Integer24* is the internal name of this value within the source code,
- *0011* are bits that are used in the frame when this value is asked for,
- *24* bits is the length of data denoted with this value,
- *24 bit integer / binary* is the description shown in the graphical interface. This is the only detail which is available to the end user.

Because other enumerations employ a very similar pattern, there is a simple parameterized interface called *EnumBase*, which all the enumerations implement and thus let the rest of the code operate upon their common nice names in the same way. Also, there is usually a static *parse* method to them. This is to convert hexadecimal or binary value representations back to strict enumeration values (e.g., "0011" to *Integer24*).

3.1.9 Graphical user interface

For graphical interactions, the newest native Java GUI framework developed by Oracle – JavaFX – is used in version 8. It provides desired logic-design separation, which is why there are two files for each window defined within the program. FXML files are XML-based markup definitions for graphical user interface descriptions – they define the design. Coupled with each FXML is a Java file called controller, which determines the behavior of respective graphical components. There are three of these tuples:

- `InitialWindowController.java` and `InitialWindow.fxml` (later refered to as *Layer 1* window),
- `FrameDialogController.java` and `FrameDialog.fxml` (*Layer 2* window hereinafter),
- `DataRecordDialogController.java` and `DataRecordDialog.fxml` (*Layer 3* window).

Additionally, third type of file – cascading style sheets – can be used to change the look of the graphics easily, but is not utilized in this project, as the default JavaFX

version 8 theme of Modena was kept in use.

An extra class definition called *ValidatedChangeListener* was implemented and is used in the GUI. This class implements the JavaFX's *ChangeListener* interface and extends it with the added functionality of custom input validation as well as alert dialog presentation. Instances of this class can then be used in place of regular *ChangeListener*s wherever user input needs to be validated and a dialog window raised upon validation failure and thus avoid numerous code duplication.

3.1.10 Command-line user interfaces

There are two modes of text-based CLI operation. The first one, called ***interactive***, targets human users who do not have graphical output at their disposal (and therefore cannot use the GUI). In this mode, the program asks the user for each value that they would be able to fill in the GUI, one at a time. There are no functional limitations to this mode and an outcome identical to the one from graphical interface can be achieved. This is handled by own implementation of Read-evaluate-print loop (REPL) in the *REPL* class. It provides an easy way of interaction and guidance to the user and a simple enough implementation on the development side.

The second one, called ***machine*** or *non-interactive*, uses the much more common *switch pattern*, where every value has a *switch* or *flag* word which denotes it. When executing the program, all the flags and respective values are written successively after the command. Because of that, using this approach becomes complicated and hard to organize as the number of parameters grows. Therefore, it is not recommended for human users and has only been included to allow the generator to be operated by non-human users, i.e., scripts. The implementation uses a third party library of Commons CLI, described in Section 3.2.1.

Many classes and even enumerations in the project override a method called *toString()*. This is a method inherited all the way from Object (the root of class hierarchy in Java) and is used whenever a text representation of any object is needed. In this case, the default behaviour is not very useful as it only concatenates the object type and address in the memory. Overriden *toString()* methods allow for more related behavior (e.g., showing hexadecimal contents in case of M-Field).

3.2 External libraries

Two third-party open source libraries have been used in the generator project. It is the Commons CLI [25] developed by Apache [26] Software Foundation and the Java Simple Serial Connector [27] library developed by Alexey Sokolov. Both libraries

come in a single *jar* archive file, which allows them to be packaged into a standalone project-wide collective archive and be executed multi-platformly.

3.2.1 Commons CLI

Apache Commons CLI [25] is a simple tool for building, managing and presenting command-line parameters and parsing text input based upon these parameters. Its abilities have been used to simplify the implementation of the non-interactive CLI (described in Section 3.1.10) and specifically in the *NonInteractiveCLI* class. There, it receives the parameter definitions and when the program is run in this mode, it parses user (human or not) input into program values.

3.2.2 jSSC

Java Simple Serial Connector (jSSC) [27] is the third party open-source library which was used for serial port communication. It allows for port discovery, data writing and reading. Its features are used throughout the *IqrfDeviceModel* class. In comparison with other serial communication libraries for Java, such as rxtx [28], it does not need platform-dependent library files (*dll*s for Windows machines, *shared objects* for Linux-based computers and so on) which would have to be unpacked from the application archive, and can therefore be packaged into one single Java application resource bundle (*jar*) along with this application.

3.3 Logging

All debugging and error-state information is logged by the generator. This is done by Java's standard *logging* package and printed to the application's standard output channel. In case of running the CLI, this is the same facility as the commanding takes place at. In case of GUI, the logging is hidden unless the program was started in graphical mode from a command line, then the logging is there to be expected as well. The information being logged contains:

- records of periodically sent telegrams, including their content and timestamp of the action,
- any communication taking place on the serial port,
- attempts to connect to a device and their results,
- errors.

A minimized sample of the log output captured while connecting to the device (i), thus triggering configuration reading (ii), changing the configuration to reflect

the next telegram's attributes (iii), sending the telegram (iv) and reading module's response (v) is shown below:

```
Mar 23, 2018 4:16:05 PM core.IqrfDeviceModel          (i)
INFO: Connected to /dev/ttyAMA0

-->   00                                               (ii)
-->   01?
<--   B05C214365870A02

-->   00                                               (iii)
-->   01:EE09214365870A02
<--   OK

-->   FF0980447A0000035000E4                           (iv)
Mar 23, 2018 4:17:00 PM core.IqrfDeviceModel
INFO: Sent FF0980447A0000035000E4 to /dev/ttyAMA0

<--   020002                                           (v)
Mar 23, 2018 4:17:00 PM core.IqrfDeviceModel
INFO: Received 0 bytes in response.
```

3.4 Data fluctuation

To further resemble real-sensor-like traffic, the generator features an option to fluctuate the payload of WM-Bus telegrams during the operation and thus simulate real-world non-fixed value measurements. Every time before such a telegram is sent, Java's built-in random number generator is used to produce a value which is then added to or substracted from (decided randomly as well) the telegram's payload. If there is only one byte in the field, it is affected directly. If there are two bytes of data, the latter one is fluctuated in full effect and the influence on the first (more significant) byte is reduced. If there are more than two bytes, only the last two are fluctuated nonetheless. The maximum change that a payload can experience is fixed to 0.5 % of its value.

To demonstrate this behavior, a test was conducted. During this experiment, two telegrams were scheduled for periodic sending by the generator. Telegram A with starting payload value of `0x5100` (20 736) and transmission period of 10 seconds, and Telegram B with the initial payload of `0x5200` (20 992) and sending period of 30 seconds. Both specifications had opted for data fluctuation. The data was then

being transmitted and received[1] continually for 20 minutes. The values, as observed by the receiving device, are plotted in Figure 3.2. Both Series A (payload values

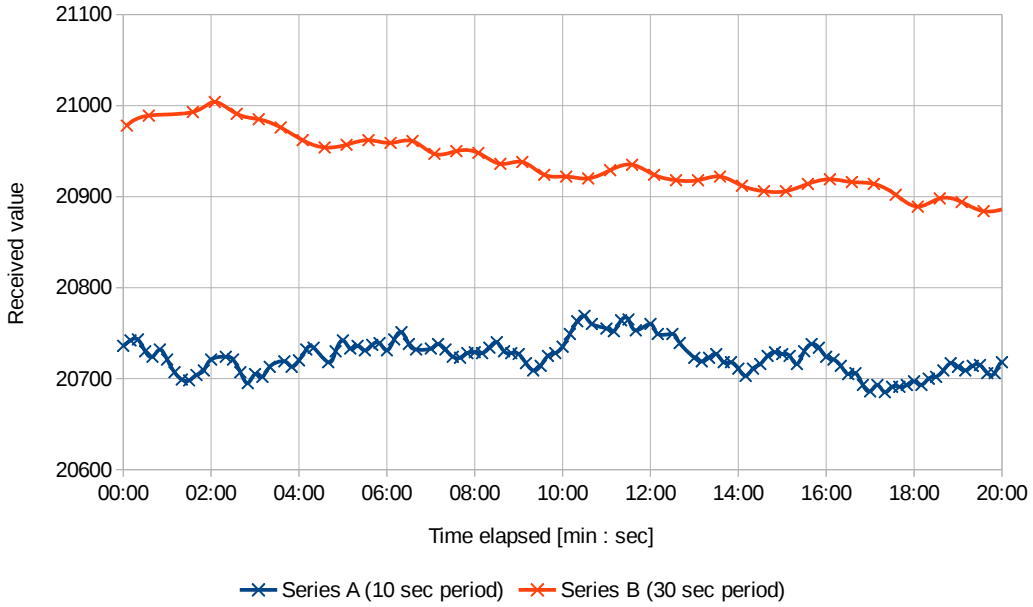


Fig. 3.2: Generated data series with simulated fluctuation.

of 120 messages produced from periodic transmiting of Telegram A) and Series B (payload values of 40 messages in Telegram B) manifest random value movements.

Log files captured and extracted from the generator (sender's perspective) and from the Amber module (receiver's perspective) can be found on the attached disc (see attachment B) in the `logs/` folder.

3.5 Documentation

Generator's project documentation was created and is stored on the attached disc (see attachment B for reference) in the `javadoc/` folder. Since it complies to the Java documentation standard called Javadoc [29], it can be browsed and viewed in the form of web pages. The entries, comments and explanations are placed in the source code files above respective methods or classes and describe the purpose of the code directly beneath. In case of methods, produced Javadoc also holds explanation on their parameters, return objects, exceptions that might be thrown (and why) and can do a lot more, which was not required in case of this project.

[1]Hardware used during test: IQRF TR-72D-WMB in UniPi Neuron S103 with an external antenna as transmitter, Amber AMB8465-M as receiver.

3.6 Unit tests

In the Java project, unit tests based on JUnit 4 are included. They reside in a standalone tests package. Methods, which it made sense with, are tested here, including most of the static utility methods and the IqrfDeviceModel library, where all reading and writing commands' functionality as well as device responsivity are verified[2]. Two basic types of tests can be found here: standard JUnit 4 tests (where class definitions start with `@RunWith(JUnit4.class)`) and special *parameterized* tests (annotated with `@RunWith(Parameterized.class)`), which also run on the JUnit framework. Regular test classes process each test method declared within and evaluate the assertions defined in them. Parameterized test classes run each test method multiple times, changing the input variables and expected outcomes. These properties are defined as *parameters* of the test class and are stored in a two-dimensional Object array. Taking advantage of this JUnit feature, it is easier to create and run test scenarios which rely on many different inputs at once, test the application behaviour under many circumstances and thus have a better and more reliable test background.

3.7 Competition comparison

Based on performed research, there are no tools, software or hardware, which could resemble the functionality of this generator. Some of similarly focused applications can control proprietary Wireless M-Bus hardware, alter its configuration and command it to send telegrams, however, none of them allows for multiple telegram periodic transmission, nor provides this large room for message configuration. Amongst them are software utilities developed by Amber to support their Wireless M-Bus radio adapters: Amber Config Center (ACC) and Amber Commander (ACM). ACC allows for simple device configuration and management, while ACM is used for WM-Bus traffic sniffing and transmitting raw data. Although these two, when combined, provide a utility able to read from / write to the hardware configuration and send bytes to the network and even offer the additional functionality for receiving and partially parsing WM-Bus traffic, they were not focused on the type of task that is achieved by the generator: powerful yet simple to use network testing tool with multiple sensor simulation using affordable hardware.

[2]In order for these tests to pass, the IQRF device has to be connected to the computer.

4 HARDWARE

This chapter details the hardware used during development of the generator and devices which the application is built to run with, most important of which is the IQRF Wireless M-Bus module described in Section 4.1 below. Options for connecting it to computers and other platforms are explained as well in Figures 4.4 and 4.5 respectively. Controlling and managing this hardware from the generator's perspective is what Section 4.6 focuses on.

4.1 IQRF WM-Bus module

This board, which has the size of a mini SIM card (commonly referred to as just SIM card), is a crucial piece of hardware, which the development was conducted on and the final product targets to. The full name is IQRF TR-72D-WMB, as found on the manufacturer's website [30]. It is an IQRF[1] transceiver, manufactured specifically for Wireless M-Bus. On the board, there are pins to allow for UART (wired) communication, 32 kB of non-volatile EEPROM memory and an embedded antenna for transmitting and receiving WM-Bus telegrams. Thanks to low power demands, this device is predetermined to being battery powered. Additionally, an external antenna with better transmit capabilities can be connected.

The UART interface is meant to provide configuration and management, and is also needed for programming and debugging purposes. It can be connected to a USB port of a computer using two other hardware devices: IQRF debugger (detailed in Section 4.1.1), and an USB to UART converter (Section 4.1.2).

The module can be operated in one of three modes, which can be cycled through freely and which differ by their usage:

- meter – acts as a subordinate device in the WM-Bus network, sends collected data,
- MUC (Multi Utility Controller), having the role of a data collector,
- sniffer – wireless traffic is monitored (observed) and can be captured (saved), but none is being created by the device itself.

For the generator's purpose, metering mode is employed.

4.1.1 IQRF debugger

An interconnecting hardware, used to convert and pass UART communication to the IQRF module. The product name is IQRF CK-USB-04A [32], detailed in Fig-

[1] IQRF is a technology for wireless packet-oriented communication via radio frequency (RF) in sub-GHz ISM bands [31] – according to Wikipedia.org.

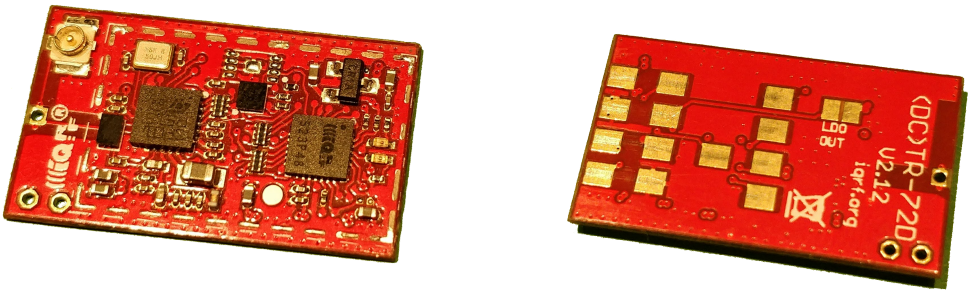
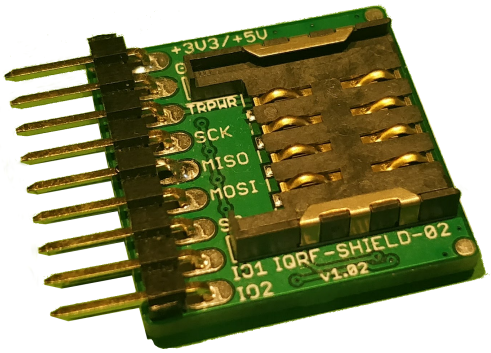


Fig. 4.1: IQRF TR-72D-WMB Wireless M-Bus module.

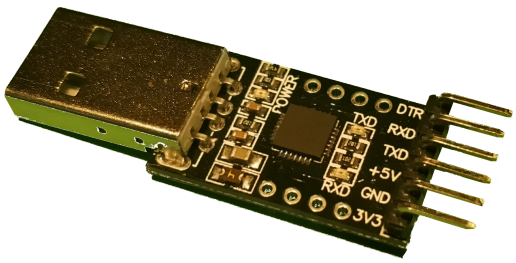
ure 4.2a. Scheme of UART pins on the board and the connections necessary to enable the communication can be seen in Figure 4.5.

4.1.2 USB to UART bridge

This is the final part of the IQRF module connection link – CP2102 Classic USB Bridge [33] manufactured by Silicon Labs. In the computer, which this converter is attached to via USB, it creates a virtual COM port interface. Any data sent from the computer to the interface are translated and transferred to the UART link. There are two LEDs, indicating ongoing data transmission (PC to UART) or data reception (UART to PC) respectively. A picture of the device is in Figure 4.2b.



(a) IQRF module debugger, CK-USB-04A.



(b) USB to UART bridge, Silicon Labs CP2102.

Fig. 4.2: Tools used to connect the IQRF module: debugger (a), bridge (b).

4.1.3 Limitations

Several constrains come from using this IQRF module and have to be taken into account or worked around.

Firstly, the device does not transmit WM-Bus telegrams simply upon sending a command with contents of the message to it. It only expects to receive part of the application layer payload in the instruction and the rest of the whole telegram (addressing, errors and many other attributes) is substituted from the module's memory. This problem was managed, solved and described in Section 4.7.

Secondly, only the encryption mode of AES with CBC and non-zero initialization vector (coded as `0x05` in WM-Bus) is supported. Because of the first outlined issue affecting the message transmission process simultaneously, this problem can not be resolved and thus other encryption modes cannot be used unless respective support is provided for the module.

Lastly, from all the Wireless M-Bus modes (described in Section 2.3) and the selection of modes the device supports, only the intersection represented by Modes S1 and S2 can be used for the generator's purpose.

4.2 Amber wireless module

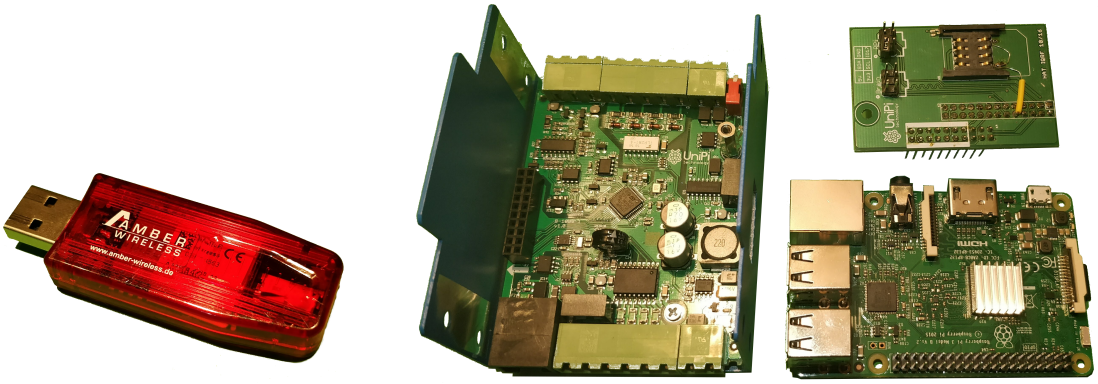
AMB8425 is a Wireless M-Bus module [34], which possesses capabilities similar to those of the IQRF module. Antenna, control chip, converters and USB port are all fitted within one casing, which seems to make it a more suitable device for the cause. However, this hardware cannot be reconfigurated or managed via third party programs and the provided software utilities only have limited capabilities.

There are two such tools: Amber Config Center (ACC) and Amber Commander (ACM), both feature a graphical user interface and both only run under Microsoft Windows OS. ACC lets the user change many of the module's parameters (current WM-Bus mode, link address, synchronization etc.), upgrade the firmware and more. ACM allows for telegram transmision and reception.

AMB8425, mostly along with ACM, was used in the development mainly for verification – it complies to Wireless M-Bus standards, which makes it a good testing receiver for telegrams sent from the generator.

4.3 UniPi control unit

UniPi Neuron S103 is the target platform for the generator, it represents an alternative to the aforementioned IQRF module – debugger – UART converter combination.



(a) Amber Wireless AMB8425.

(b) Raspbery Pi 3 (left) and IO shields (right) in UniPi Neuron S103.

Fig. 4.3: Details of Amber stick (a) and UniPi with shields (b).

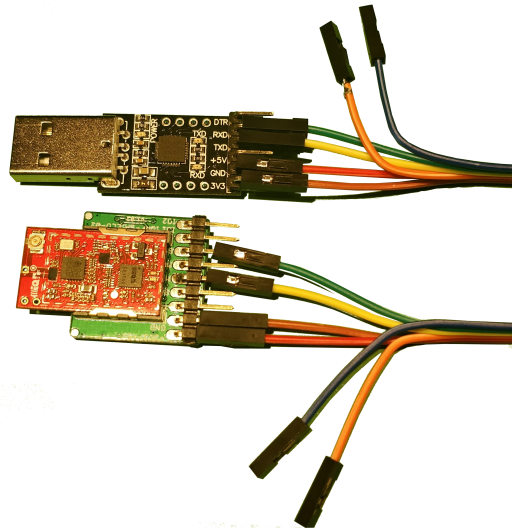
It is based on the Raspberry Pi 3 single board computer, extended with a proprietary I/O shield called UniPi and another interconnecting shield to provide more ports / options. There are multiple USB adapters, audio/video outputs as well as network card interface and it is primarily sold as a ready-to-use appliance for smart building systems, their management, regulation and monitoring [35].

This scenario requires the IQRF module (from Section 4.1) as well, it is inserted into a slot inside the UniPi boxing similar to the slot in the IQRF debugger. However, UniPi does not only serve as a more compact way of connecting the module to a computer, since it is a computer itself. The Raspberry Pi inside Neuron is powered by Raspbian – a Debian-based Linux operating system. As this is a full-fledged system, it can run Java environment, hence it can run the developed generator, too.

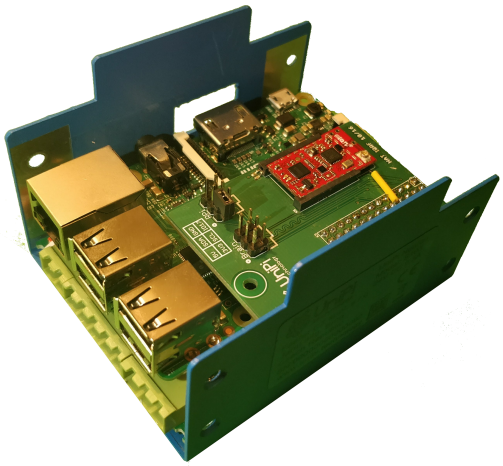
4.4 Module connection via the debugger

A simple communication link with the module can be established using the given devices. IQRF debugger (IQRF CK-USB-04A) keeps the module within its slot and mediates the connection transformation from the SIM slot to UART, which is then conducted to the UART to USB bridge (Silicon Labs CP2102). It is this device, which emulates a virtual serial port for the computer to communicate with, and thus handles most of the connection workload.

A picture of these devices interconnected is shown in Figure 4.4a and a scheme of pins connected on both sides of the UART channel is presented in Figure 4.5. In this scenario, the generator program is supposed to be run on the computer connected to the bridge.



(a) IQRF module in the debugger, connected via UART to the USB bridge.



(b) IQRF module (red) in UniPi Neuron S103.

Fig. 4.4: Approaches of connecting the IQRF module: debugger (a), UniPi (b).

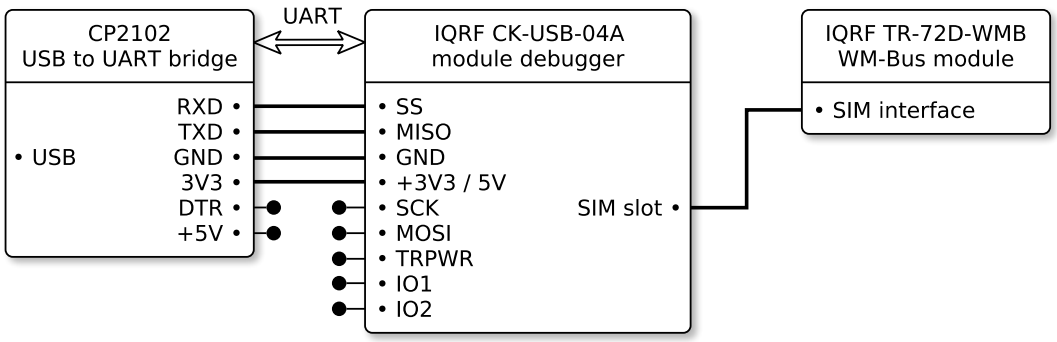


Fig. 4.5: Pin scheme of the UART interconnection.

4.5 Module connection via UniPi

To utilize the IQRF module with the UniPi control unit, the module should be inserted into respective slot, as seen in Figure 4.4b. Since this is not a standard use-case for the UniPi, there are a few configuration changes that need to be made to the Raspberry Pi board in it. Specifically, the UART interface is disabled by default and has to be activated in order to connect to the module, and, as a consequence of the previous action, serial port connected to the integrated Bluetooth antenna needs to be replaced with the serial port for the UART interface.

To perform this adjustment, following actions on configuration files of the Raspbian operating system need to be taken:

- In the file `/boot/config.txt`:
  - add (or change if present) the instruction of `ENABLE_UART=1` to enable the UART bus,
  - add the entry of `dtoverlay=pi3-miniuart-bt` to disable Bluetooth at the UART interface.
- In the file `/boot/cmdline.txt`, remove the entry of `console=ttyAMA0,115200` or similar to disable outputting debugging information to the IQRF module during system boot.
- Reboot the UniPi device.

This should ensure correct detection of the IQRF module and allow UniPi to communicate with it. Exact instructions may vary based on software versions, device types and other factors.

4.6 Controlling the module

Once a physical connection with the module has been established, whether managed through the debugger cascade or simply inserted into the UniPi slot, communication can take place. In both cases, the operating system perceives the module as a virtual serial port device with the ability of exchanging information via UART. In case of IQRF TR-72D-WMB, the interface parameteres, which are necessery to abide, are listed in Table 4.1 below. While these connection specifics are obeyed, any serial connection utility (Tera Term, HyperTerminal on Windows; screen or Minicom on Linux) should be capable of sending to and receiving from the device.

Baud rate	19200 Bd	**Parity**	none
Data bits	8	**Stop bits**	1
Flow control	none		

Tab. 4.1: IQRF TR-72D-WMB UART parameters [30].

Controlling the device can be logically devided into two parts. These are service commands (writing or reading parameters) and message transmissions (asking the device to send a WM-Bus telegram).

4.6.1 Service commands

Using service commands is necessary for performing communication targetted to the module's microchip itself, i.e., switching operation modes, changing transmission

parameters (output power, frequency), altering communication attributes (such as device's Wireless M-Bus address). The desired outcome in this case are changes to module configuration or reading from it. Several rules apply for this type of communication:

- Any service message sent **to** the device has to be preceded with a specific wakeup command (`0x00`) and a 2ms pause.
- Any service message sent **to** the device has to start with the character of ">" (`0x3E`).
- Messages received **from** the device always start with "<" (`0x3C`).
- **Every** message on the link, regardless of its direction, has to be terminated with *carriage return* character (`0x0D`).
- Commands for **writing** information use ":" (`0x3A`).
- Commands for **reading** information use "?" (`0x3F`).

This is the general format of service commands sent to the module:

```
> (0x00) (2 ms pause) [cc] [: or ?]  [data] (0x0D)
```

where parts in paratheses are fixed and not to be altered, while parts in brackets are expected to differ based on their purpose. Specifically, *cc* stands for command code (which serves as target information identifier) and *data* represent the bytes to write (if any), finished with *carriage return*.

Below is the format of device's responses:

```
< [data] (0x0D)
```

with *data* being the answer or a specific return code, again finalized with the *carriage return* character. The return codes are of three types:

- *OK* as a positive acknowledgement,
- *ERR1* to signal a syntax error and
- *ERR2* for an invalid input value.

Listed below is an example of reading the device's link address. The command code of `0x01` is used, as it corresponds with the information being asked for. The rest of the codes can be found in the official documentation [30].

```
>00                    (wake up)
>01?                   (what is your link address?)
<0EE0403020102010      (here is my link address)
```

and an example of changing it:

```
>00                     (wake up)
>01:B1257302F0849A30    (here is your new link address)
<OK                     (understood)
```

4.6.2 Message transmission

This type of communication is applied, when attempting to send a Wireless M-Bus message. Similarly to service commands, the destination of this is the module, but the action to achieve is transmitting a message wirelessly. This is done in a single – longer – command, which contains the instructions, data to be sent, their length and checksum. The message format is slightly different and relies on a different scheme, as outlined below. Also, no *carriage return* or direction pointing characters ($<$ or $>$) are used.

```
(0xFF) [length] (0x80) [data] [checksum]
```

with these rules:

- The byte of `0xFF` is mandatory and has to prepend the rest of the instruction.
- *Length* is the combined length of the message (in bytes) **after** the length byte, **including** the checksum.
- The byte of `0x80` is mandatory as well, it is the command code for sending a message.
- *Data* contain these WM-Bus components: C-Field, CI-Field, signature and payload itself the way they are meant for sending.
- *Checksum* is a simple one-byte exclusive OR (bitwise operation) of all bytes following the `0xFF` one and prioring the checksum.

Importantly, *checksum* in this message is **not** the standard CRC-Field which will be sent in the WM-Bus message. It only serves as a verification there were no errors on the UART link. Similarly, *length* does not correspond with the L-Field, but delivers the information about the instruction length to the module. The actual WM-Bus fields are calculated by the device and placed to respective positions before transmission.

The attributes of a standard WM-Bus message, which are missing in the scheme (status byte, M-Field, all components of A-Field and many others), are properties of the module itself and as such, they have to be set to the device via service commands

before requesting the transimission with a transmission command (this is covered in Section 4.7).

The module sends standard responses for these instructions, too. If there was an ansnwer expected to the message and this answer was recieved, it would be returned in the response. Other options are positive and negative acknowledgements, the latter including one of predefined error codes. General format of response messages is as follows:

```
[length] [result code] [data] [checksum]
```

where *length* is the sum of the lengths of following fields (in bytes), *result code* delivers the result of the preceding transmission, data are the response received from other, targeted Wireless M-Bus device (if any, in case of no response, this field is omitted) and an exclusive OR byte, calculated from all preceding fields.

Example of an instruction which requires the module to send a message and its response is given:

```
FF 08 80 44 7A 00 00 03 AB 1E     (instruction)
02 00 02                          (response)
```

This command asks for sending a WM-Bus message with C-field code of `0x44`, CI-field code of `0x7A`, signature of `0x(00 00)` and payload `0x(03 AB)`. Upon receiving this, the module checks the data for invalid values and errors, constructs the full telegram using values stored inside its memory, calculates the length and CRC, performs encryption, if applicable, wraps preamble and postamble around and sends it using selected WM-Bus mode.

Given by the use of unidirectional communication mode, no answer from another device was expected. Thus, only a response code was provided in the reaction, which does not conway much information: `0x02` for length, `0x00` for response (meaning the message was successfully transmitted) and `0x02` as the checksum.

4.7 Module specific parameters

Of the numerous parameters there are to Wireless M-Bus telegrams, only a few are subjects of message sending instructions as they were explained in Section 4.6.2. These can differ message to message and are therefore easy and quick to change. Specifically for the used IQRF module, this is the list of per-message parameters:

- C-Field,

- CI-Field,
- Data field, including Data header, Data record header and payload.

When asking the module to send a message, whose parameters other than these are unchaged to the last one or match current device configuration, no other action is needed. Most of the rest are not treated the same way. They are module-specific and are used in every message the device sends without being explicitly specified in the instruction. These parameters are:

- M-Field,
- A-Field (version, device type, serial number),
- Wireless M-Bus mode,
- information stored in Data header:
  - access number,
  - status code,
  - signature,
- information related to encryption:
  - encryption type to use,
  - encryption key,
  - encryption initialization vector.

Anytime a telegram is supposed to be sent and these parameters configured in the module do not match those in the telegram, the ones saved in the device have to be overwritten so that the message, when sent, corresponds with the request. In the generator, the class of *IqrfDeviceModule* is responsible for handling these adjustments.

Lastly, L-Fields and CRC-Fields are nor module-specific or included in every instruction. The module calculates them itself, before sending every message.

4.8 Performance

Albeit high performance is not a concern nor a goal for a generator of this sort, a few evaluations have been conducted in order to determine the capabilities of the application in this part of the image. Various operations were performed on the device 10 times in a row and a final average for each was recorded[2]. Following average durations, presented in Table 4.2, were observed:

[2]Configuration used: PC running Debian 9, generator running on Oracle JRE 8, USB 3.0 port, UART bridge Silabs CP2102, debugger IQRF CK-USB-04A, module IQRF TR-72D-WMB.

Operation	**Average response duration [ms]**
Initial connecting	95
Setting a single parameter	12
Writing a single parameter	8
Sending a telegram (payload 2 B)	62
Sending a telegram (payload 20 B)	71
Setting 5 parameters and sending a telegram (payload 10 B)	118

Tab. 4.2: Average response time of IQRF module for various actions.

As per the table, sending an average sized telegram and having to change 5 parameters of the device beforehand takes about 118 milliseconds. In a hypothetical yet not unrealistic scenario, where test telegrams are this long on average and where there is a need to alter approximately 5 device-specific attributes between every telegram, this generator-hardware combination would be able to send about 8.47 telegrams every second, or 30 508 telegrams per hour. Given the target usage, this is a satisfactory result, since real sensors typically only send data every few minutes at most.

5 RESULTS

Besides the broad description of the Wireless M-Bus protocol, provided in the Chapter 2, the outcome of this thesis is the Java application, developed for generating WM-Bus data in the form and structure as demanded by the user. Description of its internals was given in Chapter 3, hardware it works with in Chapter 4. A short user-side explanation follows here.

5.1 Graphical user interface

The first window presented to the user upon starting the program in the graphical mode is portrayed in the figure 5.1 below. For this reason, it will be referred to as *Layer 1* window.

5.1.1 Layer 1 window

This window shows a table of all frames that the user has defined and for each of them the interval that the frame is set to be sent in, a toggle for turning the fluctuation, an on/off switch to control whether the respective frame is scheduled to be periodically sent at the given interval or not, and a status indicator (showing whether the last frame transmission succeeded or failed). These controls are provided on a per-frame basis. For the purpose of this table, frames are presented by their link-layer content

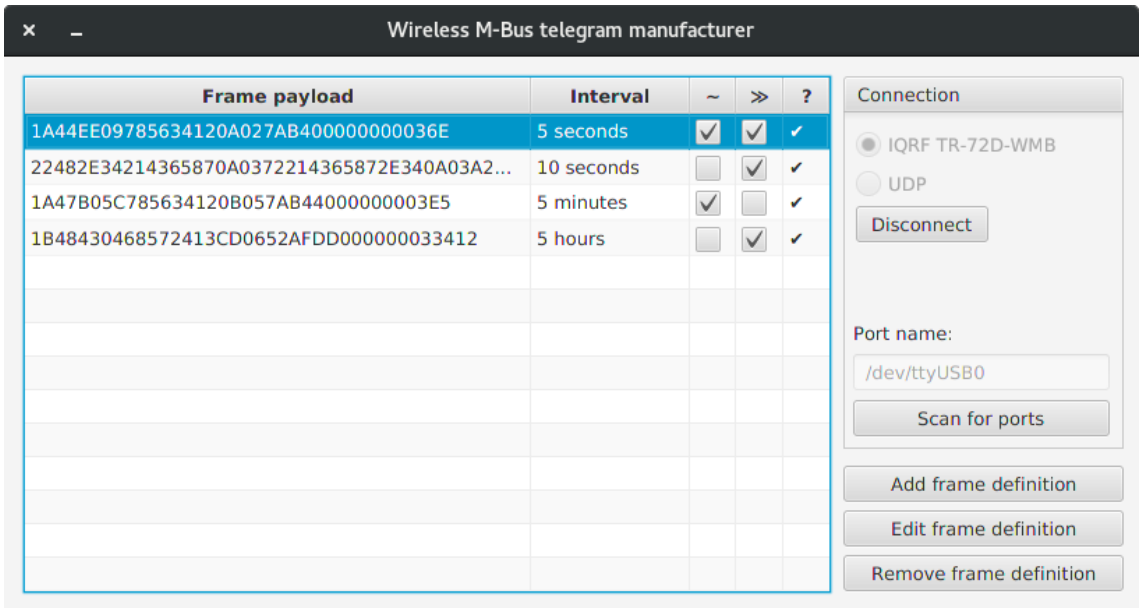


Fig. 5.1: Generator's GUI: *Layer 1* window, list of defined telegrams.

only, thus excluding preamble, postamble and any CRC fields between, and all is displayed as hexadecimal.

On the right side, global options, regarding the transmission of all frames, are placed. This is the option to switch between sending through a serial port to a supported hardware WM-Bus transceiver, or through a UDP socket (e.g., for testing). Based on this selection, more controls are presented, either to specify the serial port name or IP and UDP parameters (source and destination IP addresses and ports).

New frame definition can be added, an existing one can be edited or removed. If any type of transmission has been chosen and locked, frames in the table can be scheduled for periodic sending.

5.1.2 Layer 2 window

Upon clicking the *Add frame definition* button in *Layer 1* window, a *Layer 2* window is shown, as presented in the Figure 5.2 below.

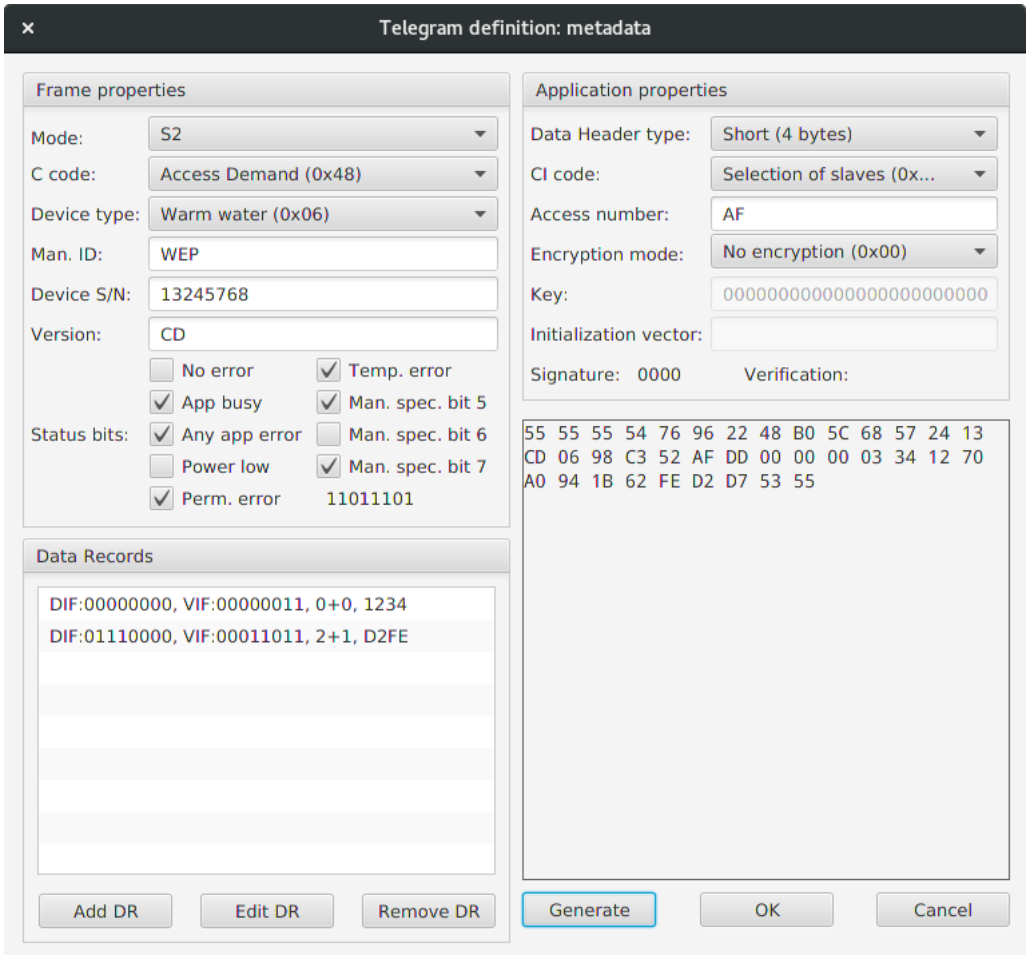


Fig. 5.2: Generator's GUI: *Layer 2* window, frame details.

The purpose of this window is to allow for definition of a new frame. All the parameters that are specific to one frame are listed here, including the manufacturer ID, C-field code, CI-field code and others alike. A list of Data Records, defined within this Frame, is presented as well. Similarly to the table in the previous window, Data Records can be added, changed and removed. Before submitting the frame, the user can generate the full byte sequence with no parts omitted, for revision.

5.1.3 Layer 3 window

Lastly, when the *Add DR* button in the *Layer 2* window is clicked, a *Layer 3* window is shown. An example of that is shown in Figure 5.3. Here, a new Data Record for the frame being specified in the underlying *Layer 2* window can be defined. This includes building a DIF, a VIF, up to ten DIFEs and VIFEs and the payload.

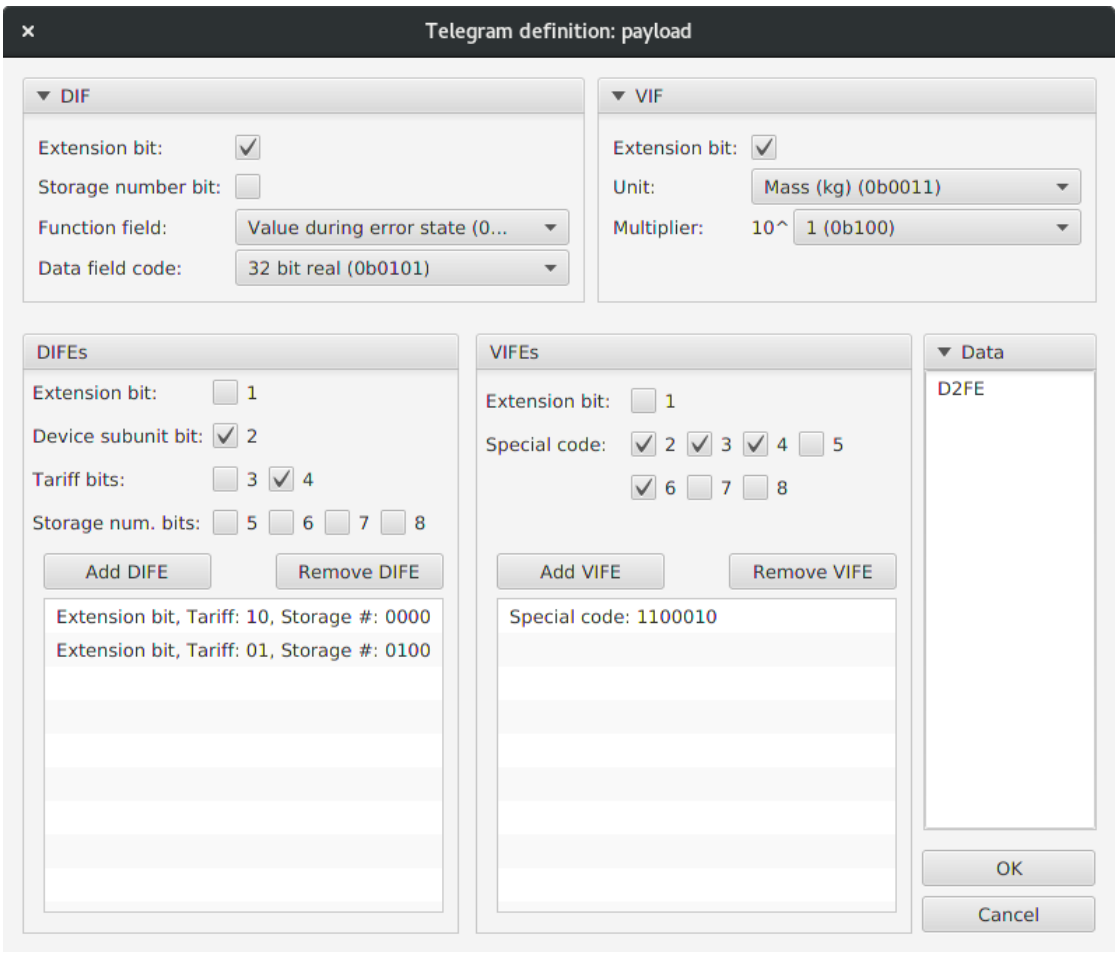


Fig. 5.3: Generator's GUI: *Layer 3* window, payload details.

This structure follows the WM-Bus hierarchy, where one frame holds one or more Data Records and each of the Data Records holds one DIF, one VIF and optionally multiple VIFEs and DIFEs, while frame itself is the only stand-alone unit. In addition, all the windows behave as modal, which means that before a lower-layer window is closed, the underlying higher-layer window cannot be acted with – for instance, a Frame definition window cannot be closed to proceed to the initial window or have its values modified until its Data Record specification windows are closed.

5.1.4 Management dialog

In the case that an IQRF module is attached to the machine and recognized by the application, the option to view a simple management window becomes available. Its appearance is portrayed in Figure 5.4. It allows for basic configuration changes in the hardware – altering the transmission power and synchronicity, restarting the device (which terminates all scheduled periodic operations and closes the connection) and views the reported battery status along with hardware and firmware version numbers.

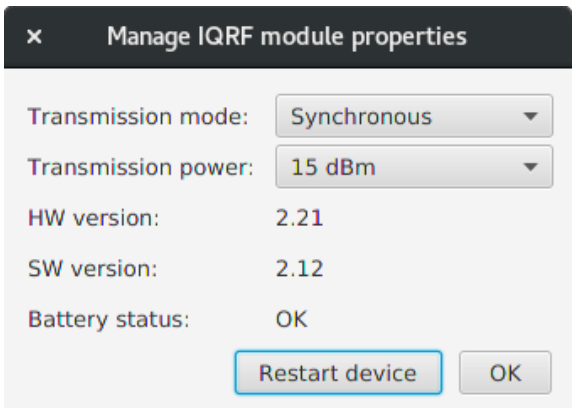


Fig. 5.4: Generator's GUI: *Management* dialog.

5.1.5 Graphical interface use-case illustration

An example of the application's use case is as following: the user starts the program. *Layer 1* window is shown with no frames in the table. The user adds a frame, customizes its properties in the *Layer 2* window, adds a Data Record with two DIFEs and two VIFEs and another Data Record without any extensions within the *Layer 3* window, closes the two upper windows and is back to the *Layer 1* window with the frame, which they have just specified, in the table. Before sending, it is neccessary to

connect to an output channel. Thus, an IQRF WM-Bus transceiver is attached via USB, the user proceeds through the connection process and the generator recognizes the hardware. The user now changes the interval to "30 seconds", opts for data fluctuation and ticks the checkbox in the column for enabling it. The frame is sent via a serial port to the transceiver, processed and wirelessly transmitted. Status tick now appears at the end of the row, signalling successful transmission. From now on, the frame will be sent out to the Wireless M-Bus network every 30 seconds [1] and more frames can be added at any time.

5.2 Command line interface

A command-line interface is included as well. It supports two modes: interactive mode targets human users, machine mode is for scripts or more advanced users. Operation of the **interactive** mode is portrayed in Figure 5.5.

```
david@edge:~/git/thesis/out(master)$ java -jar thesis.jar --interactive
---------
How do you want your telegram to be periodically sent?
1: Through UDP
2: Via serial port
2
---------
Recognized serial ports:
/dev/ttyUSB0

Enter the serial port name
/dev/ttyUSB0
Mar 30, 2018 3:08:43 PM core.IqrfDeviceModel connect
INFO: Connected to /dev/ttyUSB0
---------
Which Wireless M-Bus mode?
1: Mode S1
2: Mode S2
2
```

Fig. 5.5: Generator's CLI: interactive mode (a section of output, logging omitted).

It operates by asking the user a series of questions about necessary variables, following a similar path the user would take in the GUI. All options that the generator offers are included, as well as the definition of multiple DIFEs and VIFEs. Since the question loop does not proceed until a valid answer is provided, this is a simple way of using the CLI.

[1] Changing the interval of an enabled Frame will cause it to be sent immediately again, then further obey the new interval.

The **machine** mode works as standard single switch-based command. An example of this is provided in Figure 5.6. This mode provides only limited options of frame specification. Multiple Data Records per Frame are not supported, as it is usually very difficult to input variable numbers of arguments, in this case in many instances, to a terminal. For the same reason, no DIFEs or VIFEs are allowed. To address that, a help screen is deployed, accessible either by using --help or -h switches, or by running the application in this mode with no arguments. Part of the help output is shown in Figure 5.7. Those arguments, where a strict number of options is present (Wireless M-Bus mode, CI-field code ...) are specified via the order number of the option. Complete list of these values with their numbering is printed when `--values` switch is used.

```
david@edge:~/git/thesis/out(master)$ java -jar thesis.jar --machine --an 00 --cc
ode 1 --cf 1 --cicode 8 --data ABCD --deb --df 4 --dh 1 --dt 2 --enc 1 --fc 1 --
interval 2 --man BON --mode 1 --mult 0 --portname /dev/ttyUSB0 --sn 12345678 --s
tatus 00000000 --veb --ver 0A --vt 3
Mar 30, 2018 3:16:54 PM core.IqrfDeviceModel connect
INFO: Connected to /dev/ttyUSB0
Mar 30, 2018 3:16:54 PM core.IqrfDeviceModel writeFrame
INFO: Sent FF09804078008393ABCDC7 to /dev/ttyUSB0
Mar 30, 2018 3:16:54 PM core.IqrfDeviceModel readMessageResponse
INFO: Received 0 bytes in response.
-->   FF09804078008393ABCDC7
Mar 30, 2018 3:16:56 PM core.IqrfDeviceModel writeFrame
INFO: Sent FF09804078008393ABCDC7 to /dev/ttyUSB0
Mar 30, 2018 3:16:56 PM core.IqrfDeviceModel readMessageResponse
INFO: Received 0 bytes in response.
```

Fig. 5.6: Generator's CLI: machine mode (a section of output, logging partially omitted).

5.3 Running the generator

The application can be started in graphical mode simply by double-clicking the provided *jar* file in a graphics-enabled environment. JRE 1.8 or newer has to be installed on the target machine, including the JavaFX 8 subpackage (this usually comes together with the main JRE). If, however, command-line mode is desired, it has to be started from the terminal itself. To use the CLI, execute the following command:

```
java -jar <path to jar file> <operation mode>
```

and substitute with the Java .jar file location and the desired operation mode. These modes and respective option names are:

- `--gui`, starts the application with GUI, if that is achievable,

- `--interactive`, runs the program with human-user interactive CLI (described in Section 3.1.10),
- `--machine`, starts the generator with machine CLI (detailed in Section 3.1.10),
- `--help`, prints this list of options.

When no other option is specified, the default `--gui` switch is applied.

```
david@edge:~/git/thesis/out(master)$ java -jar thesis.jar --machine --help
For the list of item selections available in the options, use "--values".

usage: java -jar thesis.jar [--an <arg>] [--ccode <arg>] [--cf <arg>]
       [--cicode <arg>] [--data <arg>] [--deb] [--df <arg>] [--dh <arg>]
       [--dstip <arg>] [--dstport <arg>] [--dt <arg>] [--enc <arg>] [--fc
       <arg>] [--help] [--interval <arg>] [--iv <arg>] [--key <arg>]
       [--man <arg>] [--mode <arg>] [--mult <arg>] [--portname <arg>]
       [--sn <arg>] [--snb] [--srcip <arg>] [--srcport <arg>] [--status
       <arg>] [--udp] [--values] [--veb] [--ver <arg>] [--vt <arg>]
    --an <arg>          sets the access number.
    --ccode <arg>       sets the C code <value number>.
    --cf <arg>          sets the configuration field.
    --cicode <arg>      sets the CI code <value number>.
    --data <arg>        sets the data.
    --deb               sets the first DIF's extension bit.
    --df <arg>          sets the data field code <value number>.
    --dh <arg>          sets the data header type <value number>.
    --dstip <arg>       sets destination IP address in case of UDP sending.
    --dstport <arg>     sets destination port number in case of UDP
                        sending.
```

Fig. 5.7: Generator's CLI: help screen in machine mode (output reduced).

When running the generator in the machine CLI mode, this mode's specific switches and parameters have to be appended to the existing command, instead of replacing it. Furthermore, log output is visible only when the application was started from a command line (nevertheless, it *will* be visible when started from a command line in GUI mode). An example of running the generator from a terminal is given in Figure 5.8 below.

```
david@edge:~/git/thesis/out(master)$ java -jar thesis.jar --help
Usage:  --gui                   for graphical user interface. Default option.

        --interactive           for an interactive text interface.

        --machine               for a classic switch-based text interface.
                                Additional switches should be appended.

        --help                  to print this help.
```

Fig. 5.8: Generator's CLI: printing help.

6 CONCLUSION

Presented thesis pursues two main goals, both of them closely related to the communication protocol of Wireless M-Bus: to provide a broad description of the protocol, its network model, information field structure and usage areas, and to implement a multiplatform data generator utilizing the protocol to emulate the traffic from multiple sensor units. There are also two logic parts to this thesis – theoretical and practical halves.

In the **theoretical** part, which comprises of introductory explanations in Chapter 1 – *Theoretical Introduction* and Chapter 2 – *Wireless M-Bus*, the targeted protocol of Wireless M-Bus is researched and described in detail, focusing on data-link and application layers. The motivation to deploy Wireless M-Bus, its communication modes, data structure, information fields, their extensions and options for encryption are the most important topics here. A vision of the protocol's usage in the years to come and an analysis of a captured real-world protocol message are given as well.

The following **practical** part starts with describing the application (Chapter 3 – *Generator*), which was created as the primary goal of the thesis. It is a Java – hence multiplatform – program, which can generate Wireless M-Bus data from both graphical and command-line interfaces. It allows for precise protocol data unit specification and is able to send these message definitions in the form of telegrams to the Wireless M-Bus network using a supported hardware transceiver. Internal description of the software part is provided, including explanations on crucial classes, functionality, auxiliary elements and comparison with similar existing software.

Following, the hardware used along with the software part is depicted in Chapter 4 – *Hardware*. Two device variations supported by the application are a standalone Wireless M-Bus module of IQRF TR-72D-WMB and more complex solution using the UniPi Neuron S103 board. Both options are described, including available choices for connection and run scenarios. Control and management of the hardware part is also explained very closely. Albeit the issue of performance is not crucial for this solution, it has been dealt with as well.

Lastly, Chapter 5 – *Results* details the created software from user's perspective, showing the interaction interfaces and possibilities. Methods of running the application together with execution examples are provided in this chapter.

Given description of the Wireless M-Bus communication protocol, based on performed research, laid the basis for the following parts. The software-hardware combination, created as a ready-to-use generator solution in the practical part of the thesis represents a powerful option in the area of testing Wireless M-Bus networks. Thus, the goals of this thesis, as presented in the assignment specification, have been met.

BIBLIOGRAPHY

[1] CRAHMALIUC, Radu. Freight Monitoring, Industry 4.0 and Smart Grids: Main Drivers for EMEA-s IoT Spending until 2020. In: *CloudMania* [online]. 16 January 2017 [cit. 2018-05-09]. Available from: `https://cloudmania2013.com/2017/01/16/freight-monitoring-industry-4-0-and-smart-grids-main-drivers-for-emeas-iot-spending-until-2020/`

[2] ALMADA-LOBO, Francisco. Six benefits of Industrie 4.0 for businesses. In: *Control Engineering* [online]. [cit. 2018-05-09]. Available from: `https://www.controleng.com/single-article/six-benefits-of-industrie-40-for-businesses/5c57cc3925c0ff323553da64108d5c0c`

[3] BOUCHERAT, Xavier. *Industry 4.0 and the rise of smart manufacturing* [online]. 2016 [cit. 2017-10-17]. Available from: `https://www.automotiveworld.com/analysis/industry-4-0-rise-smart-manufacturing/`

[4] *Industry 4.0: The Future of Productivity and Growth in Manufacturing Industries* [online]. In: GERBERT, Philipp, Markus LORENZ a Michael RÜßMANN. 2015 [cit. 2017-10-15]. Available from: `https://www.bcg.com/en-cz/publications/2015/engineered_products_project_business_industry_4_future_productivity_growth_manufacturing_industries.aspx`

[5] VAN DER MEULEN, Rob. *Gartner Says 8.4 Billion Connected "Things" Will Be in Use in 2017, Up 31 Percent From 2016* [online]. 2017 [cit. 2017-10-15]. Available from: `https://www.gartner.com/newsroom/id/3598917`

[6] BOSWARTHICK, David. *M2M communications: a systems approach*. 1. Chichester, West Sussex, U.K.: Wiley, 2012, p. 19. ISBN 978-1-119-99475-6.

[7] VOJÁČEK, Antonín. *M-BUS (Meter-Bus): základní popis komunikačního protokolu* [online]. 2014 [cit. 2017-10-17]. Available from: `http://automatizace.hw.cz/mbus-meterbus-zakladni-popis-komunikacniho-modelu`

[8] RITTER, Terry. The Great CRC Mystery. In: *Ciphers by Ritter* [online]. [cit. 2018-05-09]. Available from: `http://www.ciphersbyritter.com/ARTS/CRCMYST.HTM`

[9] DONEV, Dimo. *Wireless M-Bus based eXtremely Low Power protocol for wireless communication with water meters*. Aalborg, 2012. Aalborg Universitet. Thesis supervisor: Neeli Prasad.

[10] NRZ Coding. In: *Vector* [online]. [cit. 2018-05-09]. Available from: https://elearning.vector.com/index.php?&wbt_ls_seite_id=522832&root=378422&seite=vl_can_introduction_en

[11] *The internet of things: key applications and protocols*. HERSENT, Olivier, David BOSWARTHICK a Omar ELLOUMI. 2012. ISBN 978-1119994350.

[12] EN 13757. *Communication systems for meters*. 2014.

[13] *Wireless M-Bus based eXtremely Low Power protocol for wireless communication with water meters* [online]. In: DONEV, Dimo. 2012, s. 13 [cit. 2017-10-17]. Available from: http://projekter.aau.dk/projekter/files/63476956/WaterMeterWSN.pdf

[14] EN 13757-4. *Communication systems for meters and remote reading of meters: Part 4: Wireless meter readout (Radio meter reading for operation in SRD bands)*. 2013.

[15] KALOUDIOTIS, Evangelos. *A 169 MHz a 868 Mhz Wireless M-Bus Based Water and Electricity Metering System*. Uppsala, 2015. Uppsala Universitet.

[16] BRUNSCHWILER, Cyrill. *Wireless M-Bus Security Whitepaper Black Hat USA 2013 June 30th, 2013* [online]. In: . 2013 [cit. 2017-10-24].

[17] Silicon Laboratories Inc. *Wireless M-Bus software implementation* [online]. [cit. 2017-10-17]. Available from: https://www.silabs.com/documents/public/application-notes/AN451.pdf

[18] Cyclic redundancy check. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2017-10-30]. Available from: https://en.wikipedia.org/wiki/Cyclic_redundancy_check

[19] Flag manufacturers ID. *DLMS User Association* [online]. [cit. 2017-10-21]. Available from: http://dlms.com/organization/flagmanufacturesids/

[20] EN 13757-3. *Communication systems for meters and remote reading of meters: Part 3: Dedicated application layer*. 2013.

[21] ZIEGLER, Horst. *Dedicated Application Layer (M-Bus)* [online]. In: . [cit. 2017-10-28]. Available from: http://www.m-bus.com/files/w4b21021.pdf

[22] ZEMAN, Kryštof *Implementace komunikačního protokolu Wireless M-BUS v simulačním prostředí NS-3*: diploma thesis. Brno: Brno University of Technology, Faculty of Electrical Engineering and Communications, Departmen of Telecommunications, 2015. 68 p. Supervised by Ing. Pavel Mašek

[23] LASSE LUETH, Knud. *Why the Internet of Things is called Internet of Things: Definition, history, disambiguation* [online]. 2014 [cit. 2017-10-29]. Available from: https://iot-analytics.com/internet-of-things-definition/

[24] EVJEN, Peder. *Wireless M-Bus breaking new ground in metering and industrial applications* [online]. [cit. 2017-12-10]. Available from: https://www.metering.com/features/wireless-m-bus-breaking-new-ground-metering-industrial-applications/

[25] *Commons CLI* [online]. [cit. 2017-11-19]. Available from: https://commons.apache.org/proper/commons-cli/

[26] *Apache Commons* [online]. [cit. 2017-11-19]. Available from: https://commons.apache.org/

[27] Official jSSC (Java Simple Serial Connector) repository. *GitHub* [online]. [cit. 2018-03-18]. Available from: https://github.com/scream3r/java-simple-serial-connector

[28] rxtx - a Java cross platform wrapper library for the serial port. *GitHub* [online]. [cit. 2018-03-18]. Available from: https://github.com/rxtx/rxtx

[29] *Oracle* - Javadoc tool. *www.oracle.com* [online]. [cit. 2018-04-08]. Available from: http://www.oracle.com/technetwork/java/javase/tech/index-jsp-135444.html

[30] IQRF *TR-72D-WMB series, Transceiver for Wireless M-Bus* [online]. [cit. 2018-03-03]. Available from: https://www.iqrf.org/products/transceivers/archive/tr-72d-wmb

[31] IQRF. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2018-03-03]. Available from: https://en.wikipedia.org/wiki/IQRF

[32] IQRF *IQRF programmer and debugger* [online]. [cit. 2018-03-03]. Available from: https://www.iqrf.org/products/ck-usb-04a

[33] Silicon Labs *CP2102 Classic USB Bridge* [online]. [cit. 2018-03-03]. Available from: https://www.silabs.com/products/interface/usb-bridges/classic-usb-bridges/device.cp2102

[34] Amber Wireless *Wireless M-Bus USB Adapter 868 MHz* [online]. [cit. 2018-03-03]. Available from: https://www.amber-wireless.com/en/amb8465-m.html

[35] UniPi Technology *UniPi Neuron S103* [online]. [cit. 2018-03-03]. Available from: `https://www.unipi.technology/unipi-neuron-s103-p93`

LIST OF ACRONYMS

ACC	Amber Config Center
ACM	Amber Commander
AES	Advanced Encryption Standard
ASCII	American Standard Code for Information Interchange
BCD	Binary-coded Decimal
CBC	Cipher Block Chaining
CLI	Command Line Interface
COSEM	Companion Specification for Energy Metering
CRC	Cyclic Redundancy Check
DAL	Dedicated Application Layer
DES	Data Encryption Standard
DH	Data Header
DIB	Data Information Block
DIF	Data Information Field
DIFE	Data Information Field Extension
DLMS	Device Language Message Specification
DRH	Data Record Header
EEPROM	Electrically Erasable Programmable Read-Only Memory
EN	European Norm
FK	Foreign key
GFSK	Gaussian Frequency Shift Keying
GUI	Graphical User Interface
HDLC	High-level Data-Link Control
IIoT	Industrial Internet of Things
IoT	Internet of Things
ISM	Industrial, Scientific and Medical
ISO	International Organization for Standardization
IV	Initialization Vector
JDBC	Java Database Connectivity
JDK	Java Development Kit
JRE	Java Runtime Environment
jSSC	Java Simple Serial Connector
JVM	Java Virtual Machine
LSB	Least-significant Bit
M2M	Machine-to-machine
MSB	Most-significant Bit
MUC	Multi Utility Controller

M-Bus	Meter Bus
NB-IoT	Narrow Band Internet of Things
NRZ	Non-Return to Zero
OSI	Open Systems Interconnection
RDBMS	Relational Database Management System
REPL	Read-evaluate-print loop
RSSI	Received Signal Strength Indicator
RTTI	Run Time Type Identification
SQL	Structured Query Language
TCP/IP	Transmission Control Protocol / Internet Protocol
UART	Universal Asynchronous Receiver Transmitter
UML	Unified Modeling Language
VIB	Value Information Block
VIF	Value Information Field
VIFE	Value Information Field Extension
WM-Bus	Wireless Meter Bus
XML	Extensible Markup Language

LIST OF APPENDICES

A CLASS DIAGRAMS

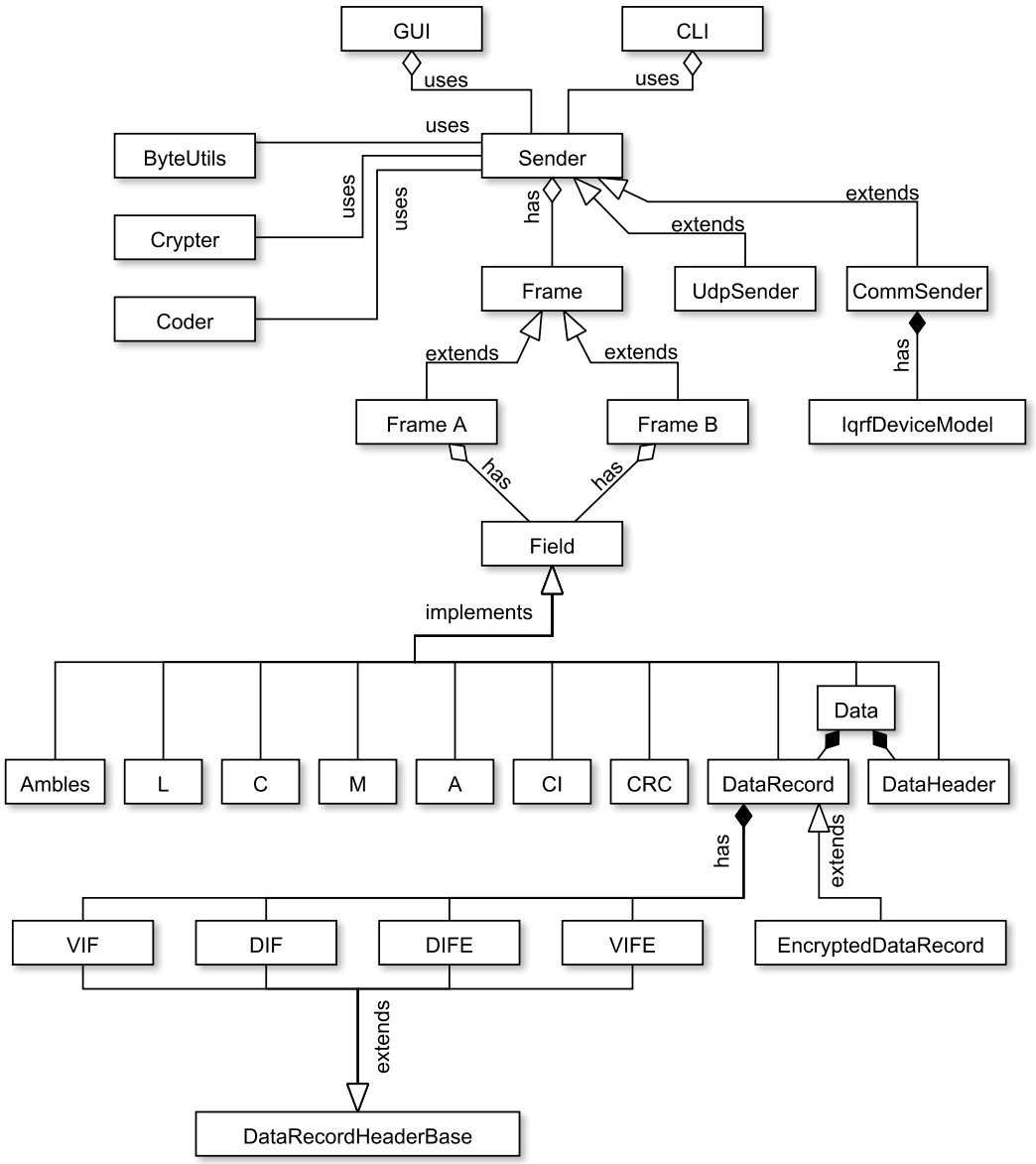


Fig. A.1: UML-like diagram for the created application's internals.

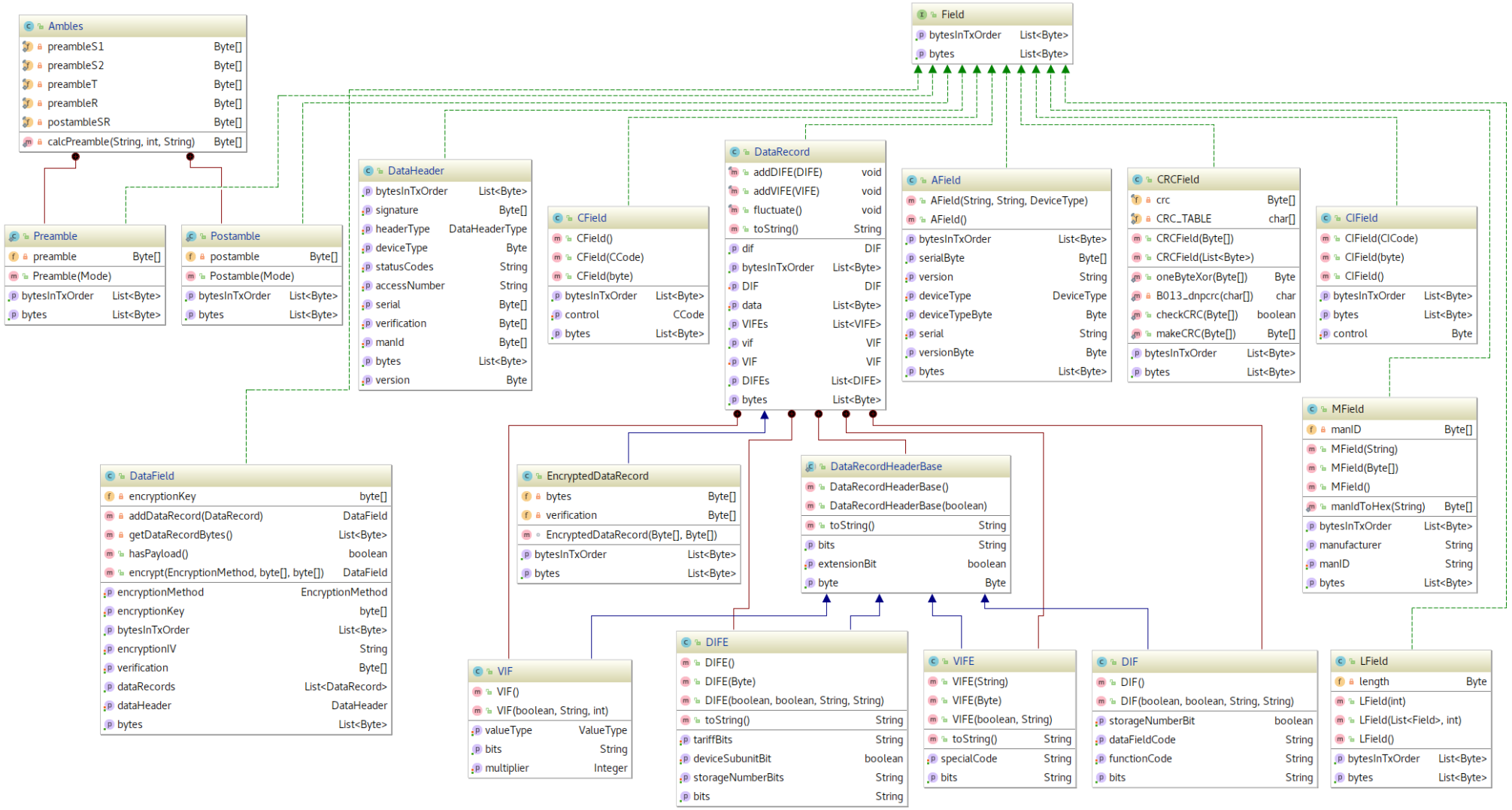


Fig. A.2: Detailed Class diagram of *Field* and its derivatives with all fields and methods.

B CONTENTS OF THE ATTACHED DISC

- `text/` — sources for thesis PDF
  - `grafy/` — chart sources
  - `loga/` — faculty logos
  - `obrazky/` — used figures
  - `pdf/` — prepared intro pages
  - `text/` — LaTeX sources
- `thesis/` — generator project
  - `lib/` — external libraries
    - `commons-cli-1.4.jar`
    - `jssc.jar`
  - `out/` — project artifacts
    - `thesis.jar` — runnable Java archive
  - `src/` — generator sources
    - `core/` — core functionality
    - `enumerations/` — enumerated types
    - `fields/` — WM-Bus fields implementations
    - `icon.png`
    - `senders/` — telegram sending functionality
    - `ui/` — sources for GUI, CLI
  - `test/` — sources for JUnit tests
- `logs/` — log files
  - `fluctuated_series_received.log` — data described in Section 3.4
  - `fluctuated_series_sent.log` — data described in Section 3.4
- `javadoc/` — Javadoc project documentation
  - `index.html` — starting page for the documentation pages
- `thesis.pdf` — the thesis in PDF
- `classdiagram.png` — full project class diagram (generated)