

# **Hardware interfaces and protocols of data exchange with Marvelmind devices**

Version 2021.11.04  
Valid for firmware v7.000 and newer

[www.marvelmind.com](http://www.marvelmind.com)

# Table of contents

1.	Connection to Marvelmind devices .....	4
1.1	UART and other interfaces for Super-Beacon .....	5
1.2	UART and other interfaces for beacon Mini-RX .....	6
1.3	UART and other interfaces for Modem HW v5.1 .....	7
1.4	UART and other interfaces for Super-Modem .....	8
1.5	UART and other interfaces for Modem HW v4.9 .....	9
1.6	UART and other interfaces for Industrial-TX, Industrial-RX, Industrial Super-Beacon... ..	10
1.7	UART and SPI interfaces for beacon HW v4.9 .....	11
1.8	UART and SPI interfaces for beacon HW v4.5 .....	12
2.	Protocols of communication via UART .....	13
2.1	'Marvelmind' protocol for streaming .....	13
2.2	Protocol of reading/writing data from/to user device .....	23
2.3	NMEA0183 communication protocol .....	27
3.	Protocols of communication via USB (virtual UART) .....	36
3.1	'Marvelmind' protocol for streaming .....	36
3.2	Protocol of reading/writing data from/to user device .....	37
3.3	NMEA0183 communication protocol .....	38
3.4	Protocol of data exchange with modem via USB interface .....	39
3.5	Marvelmind API .....	66
4.	Protocols of communication via RS-485 .....	130
4.1	'Marvelmind' protocol for streaming .....	130
4.2	Protocol of reading/writing data from/to user device .....	131
4.3	NMEA0183 communication protocol .....	132
5.	Protocols of communication via SPI .....	133
5.1	Packet with hedgehog location .....	133
5.2	Other data via SPI .....	134
6.	Protocols of communication via I <sup>2</sup> C .....	135
6.1	Compass emulation for drones with PX4 .....	135
6.2	Other data via I <sup>2</sup> C .....	136
7.	Protocols of communication via UDP (Wi-Fi) .....	137
7.1	Packet with hedgehog location .....	138
7.2	Packet with stationary beacons locations .....	139
7.3	Packet with raw IMU data .....	140
7.4	Packet with raw distances data .....	141
7.5	Packet with IMU fusion data .....	142
7.6	Packet with telemetry data .....	143

7.7	Packet with quality and extended location data.....	144
8.	Protocols of communication via CAN .....	145
8.1	'Marvelmind' protocol of streaming .....	146
8.2	NMEA0183 communication protocol .....	147
9.	Format of dashboard csv log file.....	148
9.1	Format of csv log file (dashboard version V7.000+) .....	149
9.2	Previous format of csv log (dashboard before V7.000 or modem HW v4.9) .....	159
10.	Contacts .....	160
Appendix 1. Calculating CRC-16.....		161
Appendix 2. Format of error reply from modem .....		162

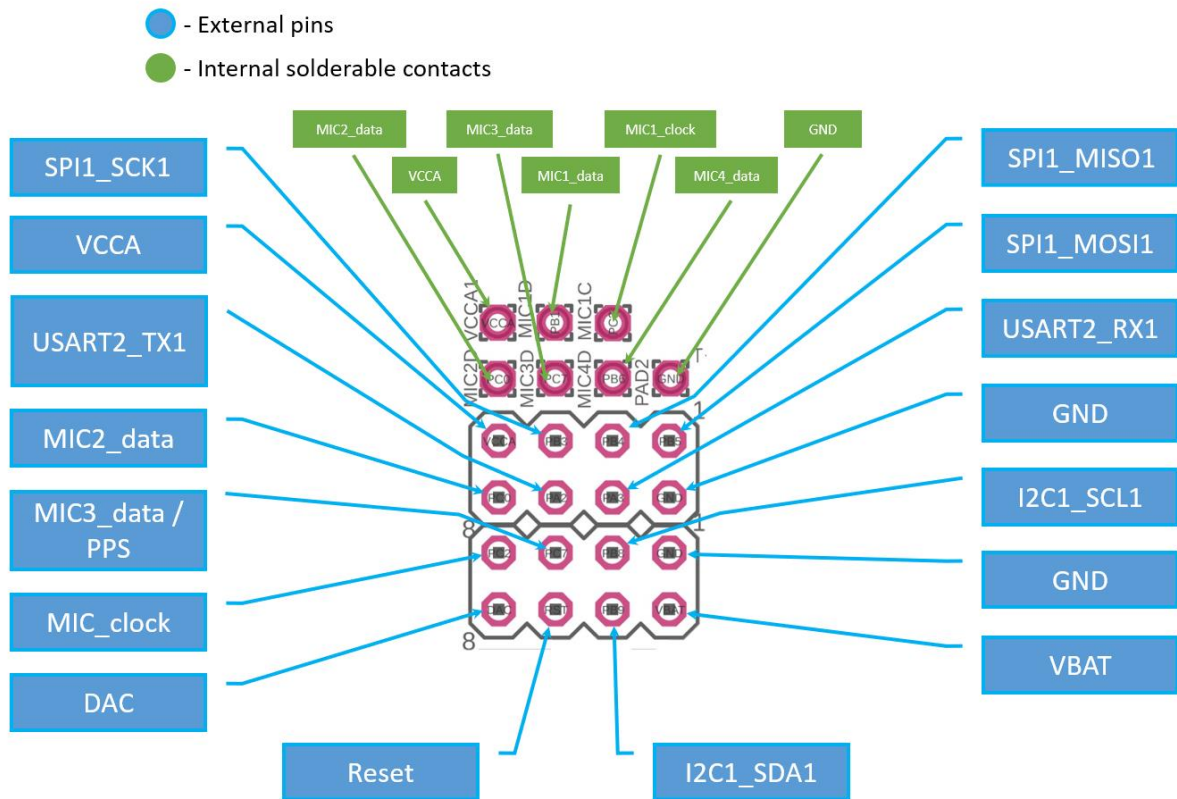
# 1. Connection to Marvelmind devices

For communication with Marvelmind devices (modem or mobile beacon (hedgehog)), it shall be connected to an external device (robot, copter, AGV, etc.) via any of the following interfaces:

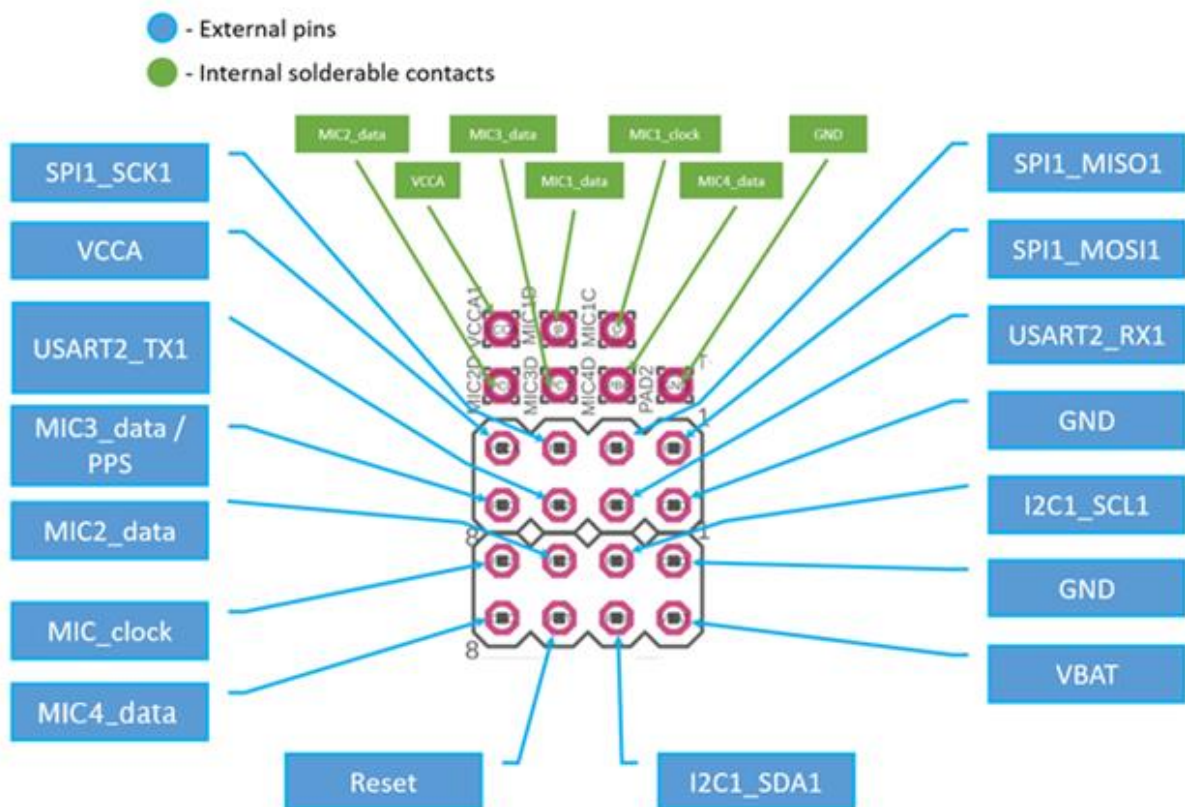
1. Connect to USB-host as an USB device of CDC class (virtual COM port in Windows, ttyACM or ttyUSB in Linux). In the Windows, it requires driver - the same driver as for modem. In Linux, the driver is not required, since the required driver is integrated into Linux kernel. Because real RS-232 is not used in the interface, parameters of serial port opened on the host (baudrate, number of bits, parity, etc) can be any.
2. Connect to UART – 2 wires soldering to pins for streaming or 3 wires for bidirectional communication required. See the picture of hardware interface below. Logic level of UART transmitter is CMOS 3.3V. Default baudrate is 500 kbps, configurable from the Dashboard from following list: 4.8, 9.6, 19.2, 38.4, 57.6, 115.2, 500 kbps. Format of data: 8 bit, no parity, 1 stop bit.
3. Connect to SPI. Marvelmind device acts as SPI slave device. Parameters of SPI: SPI mode 0, MSB inside each byte transmits first. Connection was tested on SCK speed up to 8 MHz. Be careful to provide quality wiring connections on high speeds (more than 500 kHz).
4. Connect to RS-485 (for Super-Modem or Industrial Super-Beacon only).
5. Connect to I2C (for Super-Beacon only).
6. Connect to UDP via Wi-Fi (for Super-Modem) or any network connection (for Dashboard).
7. Connect to CAN (for Industrial Super-Beacon, for Super-Modem supplied by request).

## 1.1 UART and other interfaces for Super-Beacon

4x4 pinout for Super-beacon:



4x4 pinout for Super-Beacon 2:



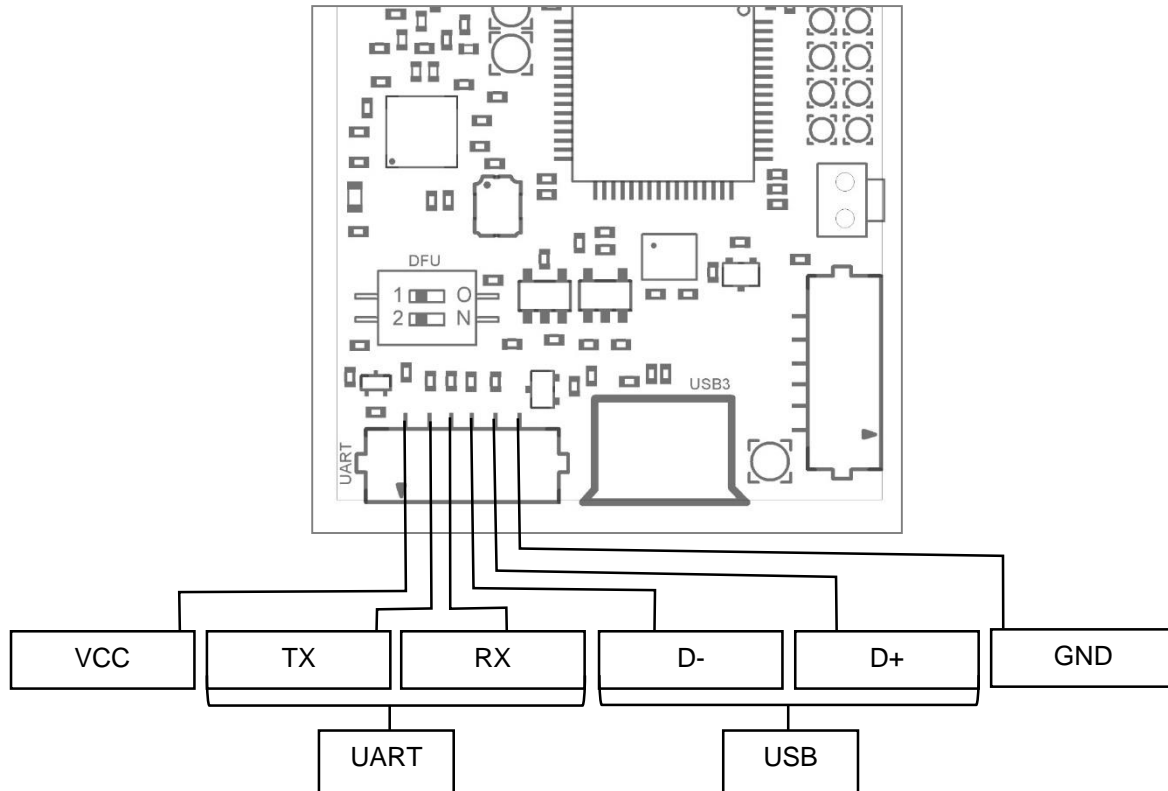
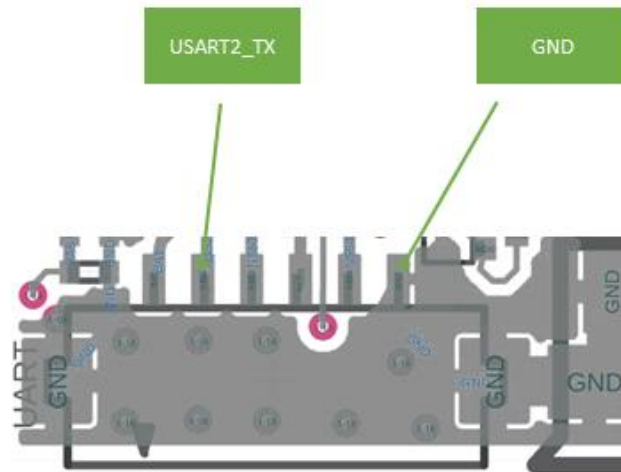
## 1.2 UART and other interfaces for beacon Mini-RX



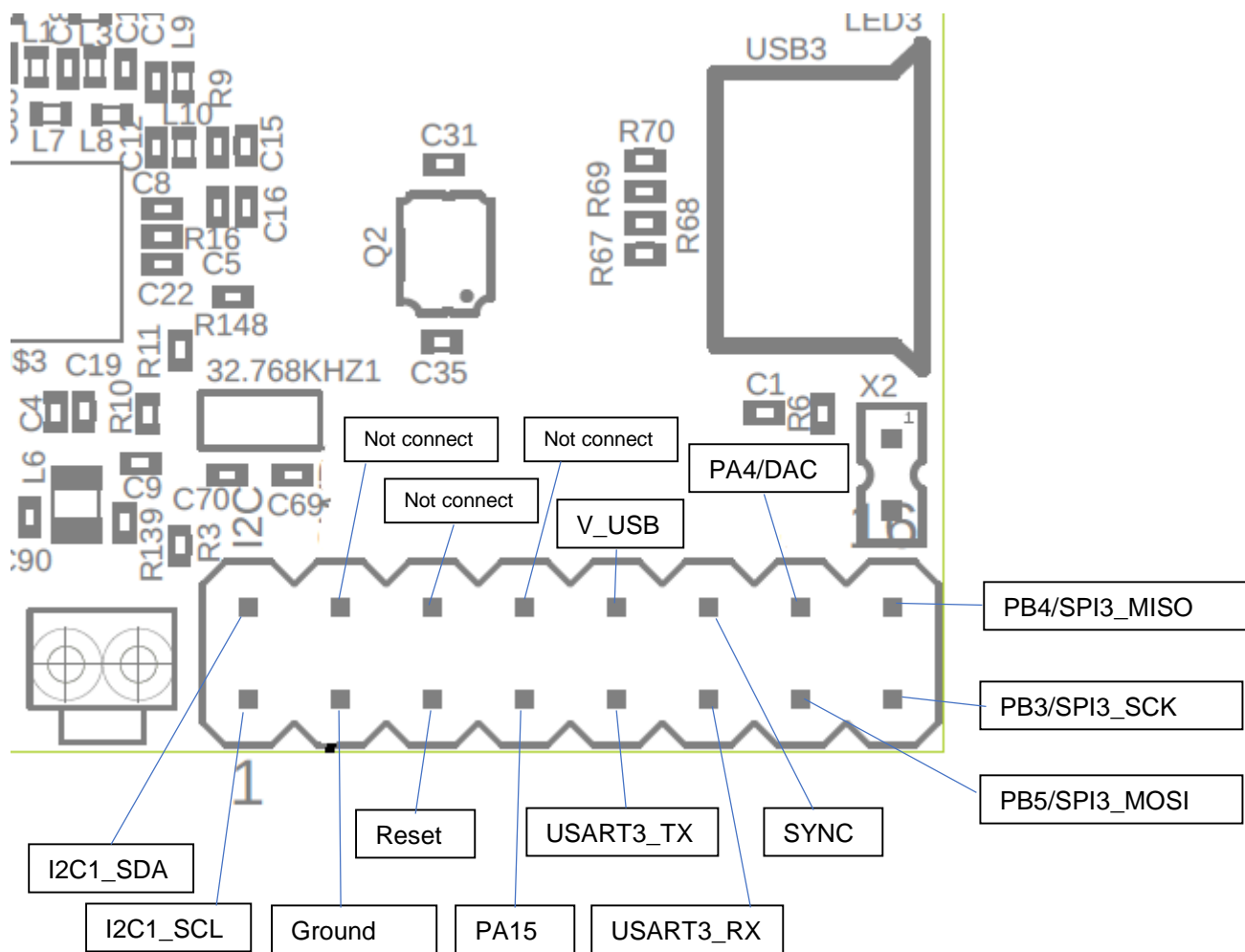
Use it only if you are sure that you can solder it correctly  
Do not forget to turn off the beacon with DIP-switches  
If you solder bad and kill the beacon, Marvelmind team won't be responsible for it

To get UART data streaming from beacon Mini-RX, you must solder to the pins on the board.

● - Internal solderable contacts

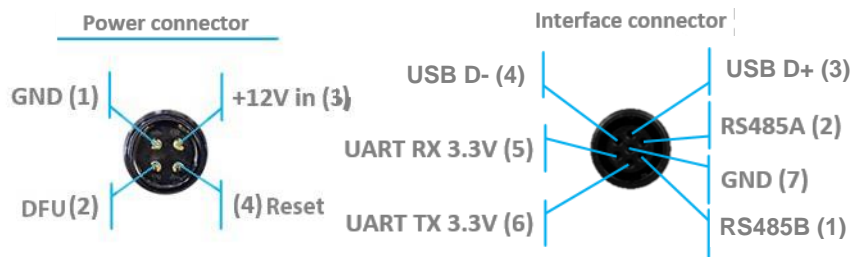


### 1.3 UART and other interfaces for Modem HW v5.1



## 1.4 UART and other interfaces for Super-Modem

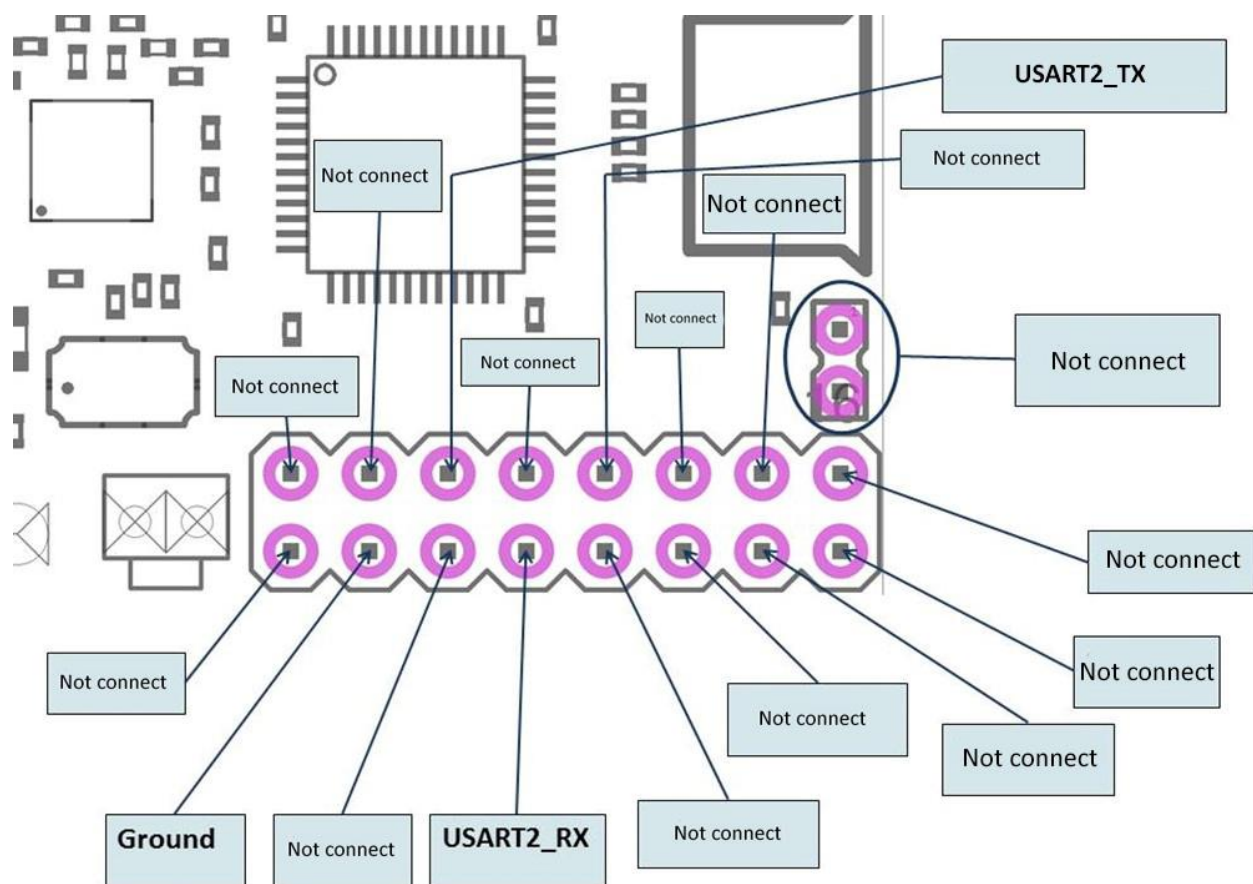
Super-Modem connectors pinout:



Also Super-Modem includes onboard WiFi interface. Configuration of the WiFi connection is described in [UDP chapter](#).

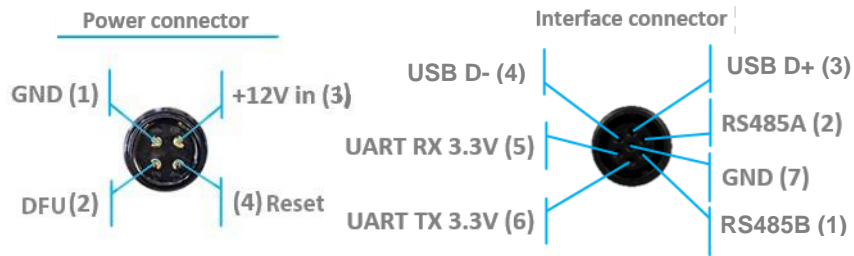


## 1.5 UART and other interfaces for Modem HW v4.9



## 1.6 UART and other interfaces for Industrial-TX, Industrial-RX, Industrial Super-Beacon

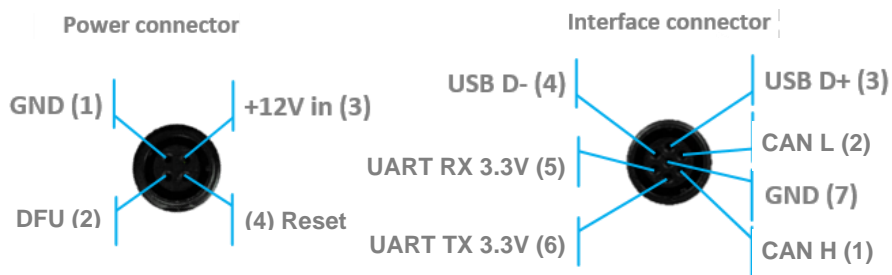
RS485 modification pinouts (After Sep.2019)



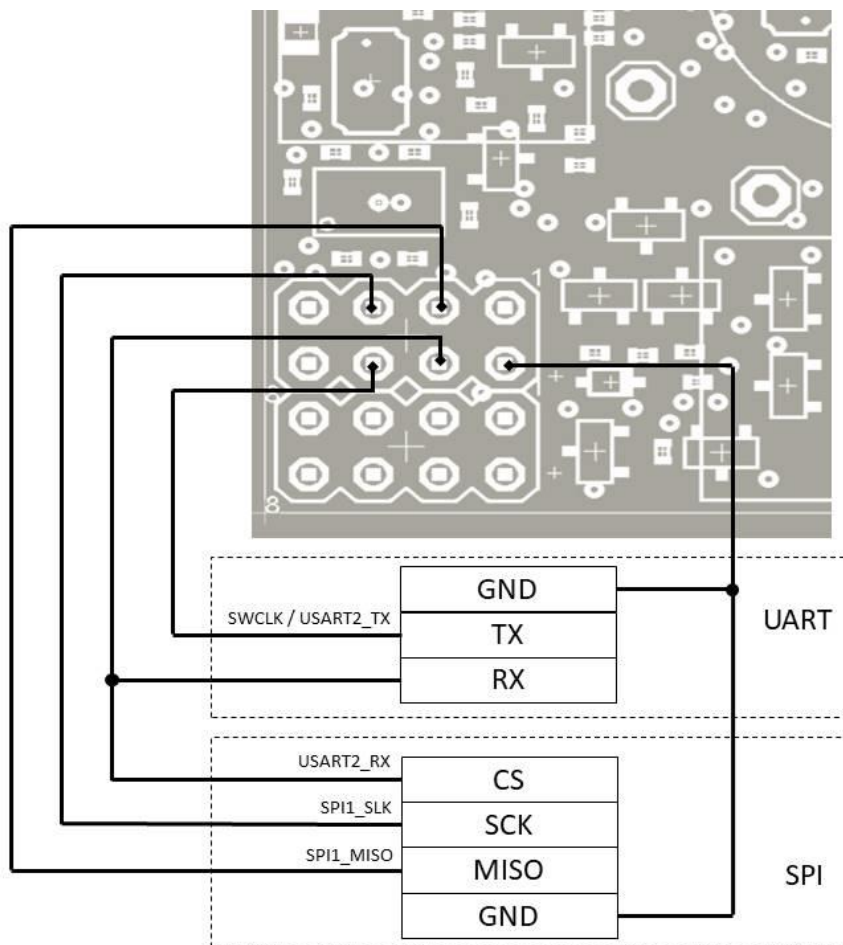
RS485 modification pinouts (Before Sep.2019)



CAN modification pinouts

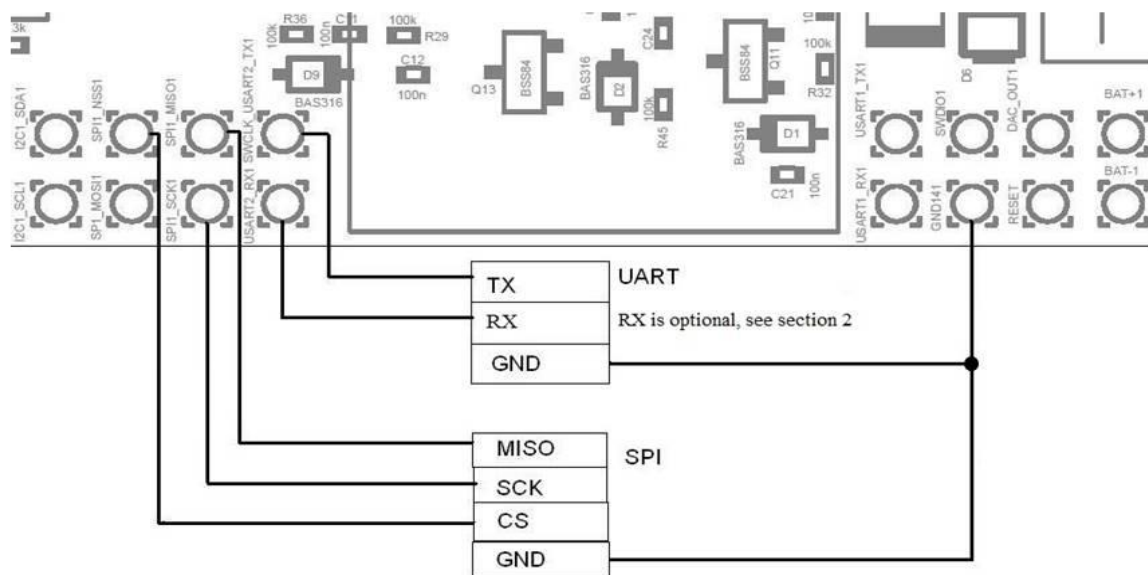


## 1.7 UART and SPI interfaces for beacon HW v4.9



Note: As you see, the UART RX and SPI CS use the same shared pin. The function of this pin (UART receiver, SPI chip select or others) can be selected in dashboard by parameter 'PA15 pin function' in 'Interfaces' section.

## 1.8 UART and SPI interfaces for beacon HW v4.5



## 2. Protocols of communication via UART

### 2.1 'Marvelmind' protocol for streaming

All streaming packets have same general structure:

Offset	Size (bytes)	Type	Description	Value
0	1	uint8_t	Destination address	0xff
1	1	uint8_t	Type of packet	0x47
2	2	uint16_t	Code of data in packet	See detail
4	1	uint8_t	Number of bytes of data transmitting	N
5	N	N bytes	Payload data according to code of data field	
5+N	2	uint16_t	CRC-16 (see appendix 1)	

## 2.1.1 Packet of hedgehog coordinates

This packet is transmitted every time new coordinates are measured or failed to measure.

### 2.1.1.1 Packet with cm resolution coordinates

#### Supported hardware:

Super-Beacon:	supported
Industrial Super-Beacon:	supported
Modem HW5.1:	supported
Super-Modem:	supported
Mini-Rx (Badge, Helmet, etc.):	supported
Mini-Tx:	not supported in the current HW version
Modem HW4.9:	supported
Beacon HW4.9:	supported
Beacon HW4.5:	supported

Offset	Size (bytes)	Type	Description	Value
0	1	uint8_t	Destination address	0xff
1	1	uint8_t	Type of packet	0x47
2	2	uint16_t	Code of data in packet	0x0001
4	1	uint8_t	Number of bytes of data transmitting	0x10
5	4	uint32_t	Timestamp – internal time of beacon ultrasound emission, in milliseconds from the moment of the latest wakeup event. See note.	
9	2	int16_t	Coordinate X of beacon, cm	
11	2	int16_t	Coordinate Y of beacon, cm	
13	2	int16_t	Coordinate Z, height of beacon, cm	
15	1	uint8_t	Byte of flags: Bit 0: 1 - coordinates unavailable. Data from fields X, Y, Z should not be used. Bit 1: timestamp units indicator (see note) Bit 2: 1 - user button is pushed (V5.23+) Bit 3: 1 - data are available for uploading to user device, see section 2 (V5.34+) Bit 4: 1 - want to download data from user device, see section 2 (V5.34+) Bit 5: 1 – second user button is pushed (V5.74+) Bit 6: 1 – data for another hedgehog (not same one that sending this packet) Bit 7: – reserved (0)	
16	1	uint8_t	Address of hedgehog	
17	2	uint16_t	Bit 0...11: orientation of hedgehogs pair in XY plane, decidegrees (0...3600) Bit 12: 1 – coordinates are given for center of beacons pair; 0 – coordinates for specified beacon Bit 13: 1 - orientation is not applicable Bit 14...15: reserved (0)	
19	2	uint16_t	Time passed from ultrasound emission to current time, milliseconds (V5.88+)	
21	2	uint16_t	CRC-16 (see appendix 1)	

### 2.1.1.2 Packet with mm resolution coordinates (firmware V5.35+)

#### Supported hardware:

Super-Beacon:	supported
Industrial Super-Beacon:	supported
Modem HW5.1:	supported
Super-Modem:	supported
Mini-Rx (Badge, Helmet, etc.):	supported with <a href="#">the UART cable for Mini-Rx</a>
Mini-Tx:	not supported in the current HW version
Modem HW4.9:	supported
Beacon HW4.9:	supported
Beacon HW4.5:	supported

Offset	Size (bytes)	Type	Description	Value
0	1	uint8_t	Destination address	0xff
1	1	uint8_t	Type of packet	0x47
2	2	uint16_t	Code of data in packet	0x0011
4	1	uint8_t	Number of bytes of data transmitting	0x16
5	4	uint32_t	Timestamp – internal time of beacon ultrasound emission, in milliseconds from the moment of the latest wakeup event. See note.	
9	4	int32_t	Coordinate X of beacon, mm	
13	4	int32_t	Coordinate Y of beacon, mm	
17	4	int32_t	Coordinate Z, height of beacon, mm	
21	1	uint8_t	Byte of flags: Bit 0: 1 - coordinates unavailable. Data from fields X,Y,Z should not be used. Bit 1: timestamp units indicator (see note) Bit 2: 1 - user button is pushed (V5.23+) Bit 3: 1 - data are available for uploading to user device, see section 2 (V5.34+) Bit 4: 1 - want to download data from user device, see section 2 (V5.34+) Bit 5: 1 – second user button is pushed (V5.74+) Bit 6: 1 – data for another hedgehog (not same one that sending this packet) Bit 7: – 1 – out of geofencing zone	
22	1	uint8_t	Address of hedgehog	
23	2	uint16_t	Bit 0...11: orientation of hedgehogs pair in XY plane, decidegrees (0...3600) Bit 12: 1 – coordinates are given for center of beacons pair; 0 – coordinates for specified hedgehog Bit 13: 1 - orientation is not applicable Bit 14...15: reserved (0)	
25	2	uint16_t	Time passed from ultrasound emission to current time, milliseconds (V5.88+)	
27	2	uint16_t	CRC-16 (see appendix 1)	

Note: for firmware versions before V5.20 timestamp is in 1/64 sec units and timestamp units indicator (bit 1 of flags byte) is 0. For versions 5.20 and higher timestamp is in milliseconds and timestamp units indicator is 1.

## 2.1.2 Packet of all beacon's coordinates (code of data 0x0002)

This packet is transmitted when the map is frozen, and repeats every 10 sec.

### 2.1.2.1 Packet with cm resolution coordinates.

#### Supported hardware:

Super-Beacon:	supported
Industrial Super-Beacon:	supported
Modem HW5.1:	supported
Super-Modem:	supported
Mini-Rx (Badge, Helmet, etc.):	supported with <a href="#">the UART cable for Mini-Rx</a>
Mini-Tx:	not supported in the current HW version
Modem HW4.9:	supported
Beacon HW4.9:	supported
Beacon HW4.5:	supported

Offset	Size (bytes)	Type	Description	Value
0	1	uint8_t	Address	0xff
1	1	uint8_t	Type of packet	0x47
2	2	uint16_t	Code of data in packet	0x0002
4	1	uint8_t	Number of bytes of data transmitting	1+N*8
5	1	uint8_t	Number of beacons in packet	N
6	1	N*8 bytes	Data for N beacons	
6+N*8	2	uint16_t	CRC-16 (see appendix 1)	

Format of data structure for every of N beacons:

Offset	Size (bytes)	Type	Description
0	1	uint8_t	Address of beacon
1	2	int16_t	Coordinate X of beacon, cm
3	2	int16_t	Coordinate Y of beacon, cm
5	2	int16_t	Coordinate Z, height of beacon, cm
7	1	uint8_t	Reserved (0)



### 2.1.2.2 Packet with mm resolution coordinates (firmware V5.35+)

#### Supported hardware:

Super-Beacon:	supported
Industrial Super-Beacon:	supported
Modem HW5.1:	supported
Super-Modem:	supported
Mini-Rx (Badge, Helmet, etc.):	supported with <a href="#">the UART cable for Mini-Rx</a>
Mini-Tx:	not supported in the current HW version
Modem HW4.9:	supported
Beacon HW4.9:	supported
Beacon HW4.5:	supported

Offset	Size (bytes)	Type	Description	Value
0	1	uint8_t	Address	0xff
1	1	uint8_t	Type of packet	0x47
2	2	uint16_t	Code of data in packet	0x0012
4	1	uint8_t	Number of bytes of data transmitting	1+N*14
5	1	uint8_t	Number of beacons in packet	N
6	1	N*14 bytes	Data for N beacons	
6+N*14	2	uint16_t	CRC-16 (see appendix 1)	

Format of data structure for every of N beacons:

Offset	Size (bytes)	Type	Description
0	1	uint8_t	Address of beacon
1	4	int32_t	Coordinate X of beacon, mm
5	4	int32_t	Coordinate Y of beacon, mm
9	4	int32_t	Coordinate Z, height of beacon, mm
13	1	uint8_t	Bit 0: 1 = location not applicable Bit 1...7: reserved

### 2.1.3 Packet of raw inertial sensors data (code of data 0x0003)

This packet is transmitted when new inertial sensors data available.

#### Supported hardware:

Super-Beacon:	supported, 100 Hz (if 'Raw inertial sensors data' enabled)
Industrial Super-Beacon:	supported, 100 Hz (if 'Raw inertial sensors data' enabled)
Modem HW5.1:	supported, system update rate (if 'IMU via modem' enabled)
Super-Modem:	supported, system update rate (if 'IMU via modem' enabled)
Mini-Rx (Badge, Helmet, etc.):	supported, 100 Hz (if 'Raw inertial sensors data' enabled) with <a href="#">the UART cable for Mini-Rx</a>
Mini-Tx:	not supported in the current HW version
Modem HW4.9:	supported, system update rate (if 'IMU via modem' enabled)
Beacon HW4.9:	supported, 100 Hz (if 'Raw inertial sensors data' enabled)
Beacon HW4.5:	supported, 100 Hz (if 'Raw inertial sensors data' enabled)

Offset	Size (bytes)	Type	Description	Value
0	1	uint8_t	Address	0xff
1	1	uint8_t	Type of packet	0x47
2	2	uint16_t	Code of data in packet	0x0003
4	1	uint8_t	Number of bytes of data transmitting	0x20
5	32		Data packet (see lower)	
37	2	uint16_t	CRC-16 (see appendix 1)	

#### Format of data packet

Offset	Size (bytes)	Type	Description	Value
0	2	int16_t	Accelerometer, X axis, 1 mg/LSB	
2	2	int16_t	Accelerometer, Y axis, 1 mg/LSB	
4	2	int16_t	Accelerometer, Z axis, 1 mg/LSB	
6	2	int16_t	Gyroscope, X axis, 0.0175 dps/LSB	
8	2	int16_t	Gyroscope, Y axis, 0.0175 dps/LSB	
10	2	int16_t	Gyroscope, Z axis, 0.0175 dps/LSB	
12	2	int16_t	Compass, X axis, 1100 LSB/Gauss	
14	2	int16_t	Compass, Y axis, 1100 LSB/Gauss	
16	2	int16_t	Compass, Z axis, 980 LSB/Gauss	
18	1	uint8_t	Address of beacon	
19	5	5 bytes	Reserved (0)	
24	4	uint32_t	Timestamp, ms	
28	1	uint8_t	Flags: Bit 0: 1 = accelerometer data n/a Bit 1: 1 = Gyroscope data n/a Bit 2: 1 = Compass data n/a Bit 3...7 – reserved (0)	
29	3	3 bytes	reserved	

Note: Compass data are available only for HW v4.9 beacons with IMU.

### 2.1.4 Packet of raw distances data (code of data 0x0004)

This packet is transmitted every time new coordinates are measured or failed to measure, after the packet with coordinates (code 0x0001/0x0011).

Available only if “raw distances data” option is enabled in ‘Interfaces’ section of settings.

#### Supported hardware:

Super-Beacon:	supported
Industrial Super-Beacon:	supported
Modem HW5.1:	supported
Super-Modem:	supported
Mini-Rx (Badge, Helmet, etc.):	supported with <a href="#">the UART cable for Mini-Rx</a>
Mini-Tx:	not supported in the current HW version
Modem HW4.9:	supported
Beacon HW4.9:	supported
Beacon HW4.5:	supported

Offset	Size (bytes)	Type	Description	Value
0	1	uint8_t	Address	0xff
1	1	uint8_t	Type of packet	0x47
2	2	uint16_t	Code of data in packet	0x0004
4	1	uint8_t	Number of bytes of data transmitting	0x20
5	32		Data packet (see lower)	
37	2	uint16_t	CRC-16 (see appendix 1)	

#### Format of data packet

Offset	Size (bytes)	Type	Description	Value
0	1	uint8_t	Address of hedgehog	
1	6		Distance item 1	
7	6		Distance item 2	
13	6		Distance item 3	
19	6		Distance item 4	
25	4	uint32_t	Timestamp – internal time of beacon ultrasound emission, in milliseconds from the moment of the latest wakeup event (V5.89+).	
29	2	uint16_t	Time passed from ultrasound emission to current time, milliseconds (V5.89+)	
31	1	uint8_t	reserved	

#### Format of distance item

Offset	Size (bytes)	Type	Description	Value
0	1	uint8_t	Address of beacon (0 if item not filled)	
1	4	uint32_t	Distance to the beacon, mm	
5	1	uint8_t	Bit 0: 1 = Distance not applicable Bit 1...7: Reserved (0)	

## 2.1.5 Packet of processed IMU data (code of data 0x0005)

This packet is transmitted when new inertial sensors data available.

### Supported hardware:

Super-Beacon:	supported, 100 Hz (if 'Processed IMU data' enabled)
Industrial Super-Beacon:	supported, 100 Hz (if 'Processed IMU data' enabled)
Modem HW5.1:	supported, system update rate (if 'IMU via modem' enabled)
Super-Modem:	supported, system update rate (if 'IMU via modem' enabled)
Mini-Rx (Badge, Helmet, etc.):	supported, 100 Hz (if 'Processed IMU data' enabled) with <a href="#">the UART cable for Mini-Rx</a>
Mini-Tx:	not supported in the current HW version
Modem HW4.9:	supported, system update rate (if 'IMU via modem' enabled)
Beacon HW4.9:	supported, 100 Hz (if 'Processed IMU data' enabled)
Beacon HW4.5:	supported, 100 Hz (if 'Processed IMU data' enabled)

Offset	Size (bytes)	Type	Description	Value
0	1	uint8_t	Address	0xff
1	1	uint8_t	Type of packet	0x47
2	2	uint16_t	Code of data in packet	0x0005
4	1	uint8_t	Number of bytes of data transmitting	0x2a
5	42		Data packet (see lower)	
47	2	uint16_t	CRC-16 (see appendix 1)	

### Format of data packet

Offset	Size (bytes)	Type	Description	Value
0	4	int32_t	Coordinate X of beacon (fusion), mm	
4	4	int32_t	Coordinate Y of beacon (fusion), mm	
8	4	int32_t	Coordinate Z of beacon (fusion), mm	
12	2	int16_t	W field of rotation quaternion	
14	2	int16_t	X field of rotation quaternion	
16	2	int16_t	Y field of rotation quaternion	
18	2	int16_t	Z field of rotation quaternion	
20	2	int16_t	Velocity X of beacon (fusion), mm/s	
22	2	int16_t	Velocity Y of beacon (fusion), mm/s	
24	2	int16_t	Velocity Z of beacon (fusion), mm/s	
26	2	int16_t	Acceleration X of beacon, mm/s <sup>2</sup>	
28	2	int16_t	Acceleration Y of beacon, mm/s <sup>2</sup>	
30	2	int16_t	Acceleration Z of beacon, mm/s <sup>2</sup>	
32	1	uint8_t	Address of beacon	
33	1	1 byte	Reserved (0)	
34	4	uint32_t	Timestamp, ms	
38	1	uint8_t	Flags: Bit 0: 1 = Location data n/a Bit 1: 1 = Quaternion data n/a Bit 2: 1 = Velocity data n/a Bit 3: 1 = Acceleration data n/a Bit 4...7 – reserved (0)	
39	3	3 bytes	Reserved (0)	

Note: Quaternion is normalized to 10000 value.

## 2.1.6 Packet of telemetry data (code of data 0x0006)

This packet is transmitted after location update, if the option “Telemetry stream” is enabled.

### Supported hardware:

Super-Beacon:	supported
Industrial Super-Beacon:	supported
Modem HW5.1:	supported (firmware V7.0+)
Super-Modem:	supported (firmware V7.0+)
Mini-Rx (Badge, Helmet, etc.):	supported with <a href="#">the UART cable for Mini-Rx</a>
Mini-Tx:	not supported in the current HW version
Modem HW4.9:	not supported
Beacon HW4.9:	supported
Beacon HW4.5:	supported

Offset	Size (bytes)	Type	Description	Value
0	1	uint8_t	Address	0xff
1	1	uint8_t	Type of packet	0x47
2	2	uint16_t	Code of data in packet	0x0006
4	1	uint8_t	Number of bytes of data transmitting	0x10
5	16		Data packet (see lower)	
21	2	uint16_t	CRC-16 (see appendix 1)	

### Format of data packet

Offset	Size (bytes)	Type	Description	Value
0	2	uint16_t	Battery voltage, mV	
2	1	int8_t	RSSI, dBm	
3	1	uint8_t	Address of the beacon	
4	12		Reserved (0)	

## 2.1.7 Packet of quality and extended location data (code of data 0x0007)

This packet is transmitted after location update, if the option “Quality and extended location data” is enabled.

### Supported hardware:

Super-Beacon:	supported
Industrial Super-Beacon:	supported
Modem HW5.1:	supported
Super-Modem:	supported
Mini-Rx (Badge, Helmet, etc.):	supported with <a href="#">the UART cable for Mini-Rx</a>
Mini-Tx:	not supported in the current HW version
Modem HW4.9:	supported (only quality field)
Beacon HW4.9:	supported (only quality field)
Beacon HW4.5:	supported (only quality field)

Offset	Size (bytes)	Type	Description	Value
0	1	uint8_t	Address	0xff
1	1	uint8_t	Type of packet	0x47
2	2	uint16_t	Code of data in packet	0x0007
4	1	uint8_t	Number of bytes of data transmitting	0x10
5	16		Data packet (see lower)	
21	2	uint16_t	CRC-16 (see appendix 1)	

### Format of data packet

Offset	Size (bytes)	Type	Description	Value
0	1	uint8_t	Device address	
1	1	uint8_t	Positioning quality, %	
2	1	uint8_t	0 = no geofencing zone alarm 1...255 - index of geofencing zone	
3	13		Reserved (0)	

## 2.2 Protocol of reading/writing data from/to user device

### 2.2.1 Sending data from user device

#### Supported hardware:

Super-Beacon:	supported
Industrial Super-Beacon:	supported
Modem HW5.1:	on demand
Super-Modem:	on demand
Mini-Rx (Badge, Helmet, etc.):	supported with <a href="#">the UART cable for Mini-Rx</a>
Mini-Tx:	not supported in the current HW version
Modem HW4.9:	not supported
Beacon HW4.9:	supported
Beacon HW4.5:	supported

If the user device needs to transmit data via Marvelmind system, it should send following frame:

Offset	Size (bytes)	Type	Description	Value
0	1	uint8_t	Destination address	0x00
1	1	uint8_t	Type of packet	0x49
2	2	uint16_t	Code of data in packet	0x0200
4	1	uint8_t	Number of bytes of data transmitting	N
5	N	N bytes	Payload data	
5+N	2	uint16_t	CRC-16 (see appendix 1)	

The data will be transmitted via radio to the modem by the parts of the size defined as 'User payload data size' in 'Interfaces' section of dashboard settings for hedgehog. The rate of sending these parts is equal to update rate of hedgehog. Buffer size in hedgehog is 128 bytes. Take this in attention to avoid overflow the buffer.

## 2.2.2 Writing data from to user device

This packet is transmitted from Marvelmind device if it wants to send data to user device.

Offset	Size (bytes)	Type	Description	Value
0	1	uint8_t	Destination address	0xff
1	1	uint8_t	Type of packet	0x4a
2	2	uint16_t	Code of data in packet	0x0200... 0x02ff
4	1	uint8_t	Number of bytes of data transmitting	N
5	N	N bytes	Payload data	
5+N	2	uint16_t	CRC-16 (see appendix 1)	

For this command the codes of data from 0x200 to 0x2ff are reserved.

If the user device successfully processed the request, it should send a response in following format:

Offset	Size (bytes)	Type	Description	Value
0	1	uint8_t	Address of hedgehog (can get from 0x0001 or 0x0011 packet of streaming)	
1	1	uint8_t	Type of packet	0x4a
2	2	uint16_t	Code of data in packet	0x0200... 0x02ff
4	2	uint16_t	CRC-16 (see appendix 1)	

If the user device failed to process the request, it sends response in following format:

Offset	Size (bytes)	Type	Description	Value
0	1	uint8_t	Address of hedgehog (get from 0x0001 packet of streaming)	
1	1	uint8_t	Type of packet	0xca
2	2	uint16_t	Code of requested data	0x0200... 0x02ff
4	1	uint8_t	Code of error (see note)	1
5	2	uint16_t	CRC-16 (see appendix 1)	

In the following sections described the specific data writing requests.

**Note:** If user device could not process request from hedgehog, it should send reply with one of following error codes:

- 1 - unknown field "type of packet" in request
- 2 - unknown field "code of data" in request
- 3 - incorrect payload data in request
- 6 - device is busy and cannot retrieve requested data now



### 2.2.2.1 Request of writing the movement path

#### Supported hardware:

Super-Beacon:	supported
Industrial Super-Beacon:	supported
Modem HW5.1:	on demand
Super-Modem:	on demand
Mini-Rx (Badge, Helmet, etc.):	supported with <a href="#">the UART cable for Mini-Rx</a>
Mini-Tx:	not supported in the current HW version
Modem HW4.9:	not supported
Beacon HW4.9:	supported
Beacon HW4.5:	supported

This packet contains one command of elementary movement. The Marvelmind device sends one after another all commands for elementary movements in the path.

Offset	Size (bytes)	Type	Description	Value
0	1	uint8_t	Destination address	0xff
1	1	uint8_t	Type of packet	0x4a
2	2	uint16_t	Code of data in packet	0x201
4	1	uint8_t	Number of bytes of data transmitting	0x0c
5	12	12 bytes	Payload data	
17	2	uint16_t	CRC-16 (see appendix 1)	

Format of payload data:

Offset	Size (bytes)	Type	Description	Value
0	1	uint8_t	Type of elementary movement: 0 - move forward 1 - move backward 2 - rotate right (clockwise) 3 - rotate left (counterclockwise) 4 - pause 5 - repeat program from start 6 - move to specified point 7 - setup speed	
1	1	uint8_t	Index of this elementary movement (0 is the first)	
2	1	uint8_t	Total number of elementary movements	
3	2	int16_t	Parameter of movement: Types 0; 1 - distance of movement, cm Types 2; 3 - angle of rotation, degrees Type 4: time of pause, ms Type 6: X target coordinate, cm Type 7: speed, %	
5	2	int16_t	Parameter of movement: Type 6: Y target coordinate, cm	
7	2	int16_t	Parameter of movement: Type 6: Z target coordinate, cm	
9	3	3 bytes	Reserved (0)	

### 2.2.2.2 Request of writing zones

#### Supported hardware:

Super-Beacon:	supported
Industrial Super-Beacon:	supported
Modem HW5.1:	on demand
Super-Modem:	on demand
Mini-Rx (Badge, Helmet, etc.):	supported with <a href="#">the UART cable for Mini-Rx</a>
Mini-Tx:	not supported in the current HW version
Modem HW4.9:	not supported
Beacon HW4.9:	supported
Beacon HW4.5:	supported

This packet contains one item of sequence of zones list. The Marvelmind device sends one after another all commands for zones list.

Offset	Size (bytes)	Type	Description	Value
0	1	uint8_t	Destination address	0xff
1	1	uint8_t	Type of packet	0x4a
2	2	uint16_t	Code of data in packet	0x202
4	1	uint8_t	Number of bytes of data transmitting	0x25
5	37	37 bytes	Payload data	
42	2	uint16_t	CRC-16 (see appendix 1)	

Format of payload data:

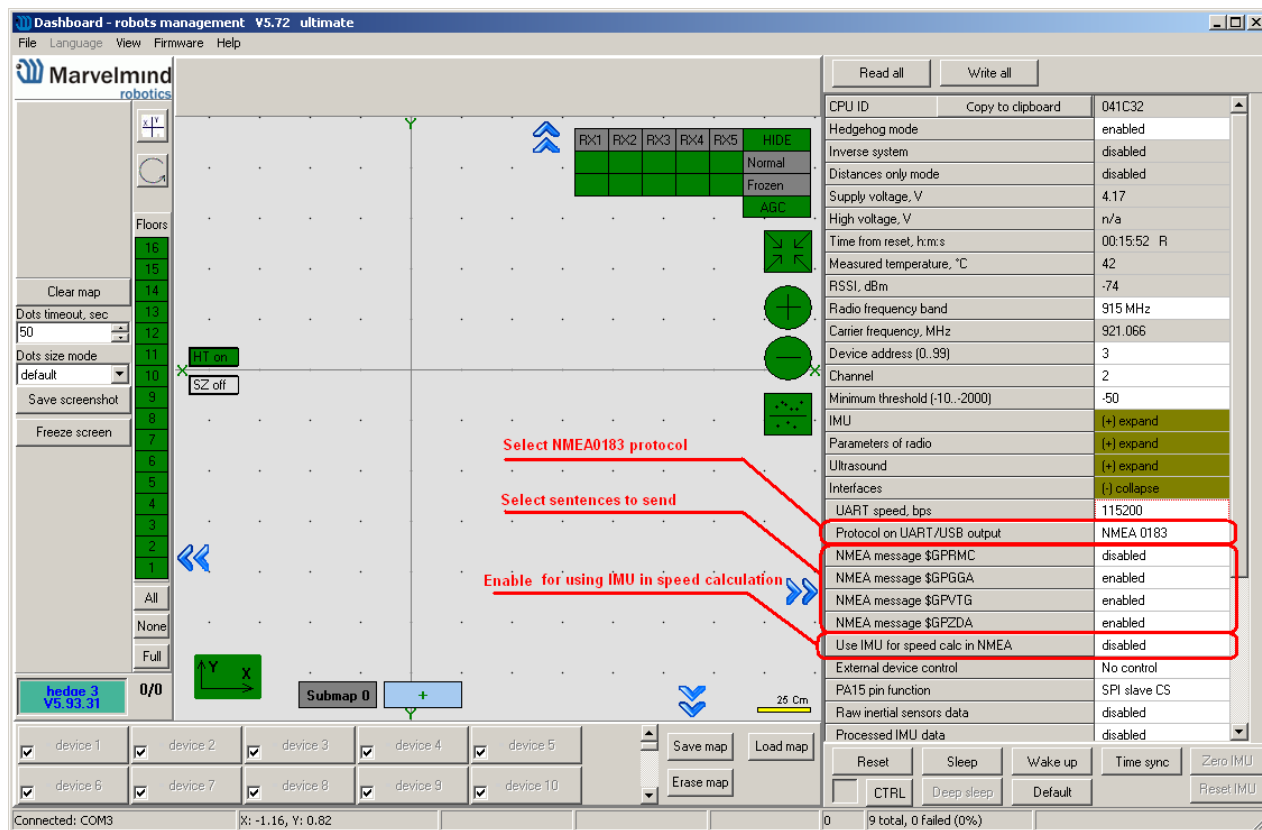
Offset	Size (bytes)	Type	Description	Value
0	1	uint8_t	Index of the zone	
1	1	uint8_t	Number of points in zone polygon (N)	
2	1	uint8_t	Index of first point in this packet: M=0...N-1	
3	1	uint8_t	Flags: Bit 0: 1 = no service zone Bit 1: 1= no driving zone Bit 2: 1= inverted zone Bit 3: 1= active zone Bit 4...7: reserved (0)	
4	1	uint8_t	Number of zones	
5	32	4x8 bytes	Up to 4 points of zone polygon (see below)	

Format of payload data:

Offset	Size (bytes)	Type	Description	Value
0	4	int32_t	X coordinate of the point, mm	
4	4	int32_t	Y coordinate of the point, mm	

## 2.3 NMEA0183 communication protocol

Mobile beacon can output some of the NMEA0183 sentences via UART and USB (virtual UART) interfaces. NMEA protocol should be enabled in the device with dashboard as shown on following screenshot:



The device sends all enabled messages every time it receives updated position.

To get NMEA data from mobile beacon (hedgehog), it shall be connected to an external device (robot, copter, AGV, etc.) via any of the following interfaces:

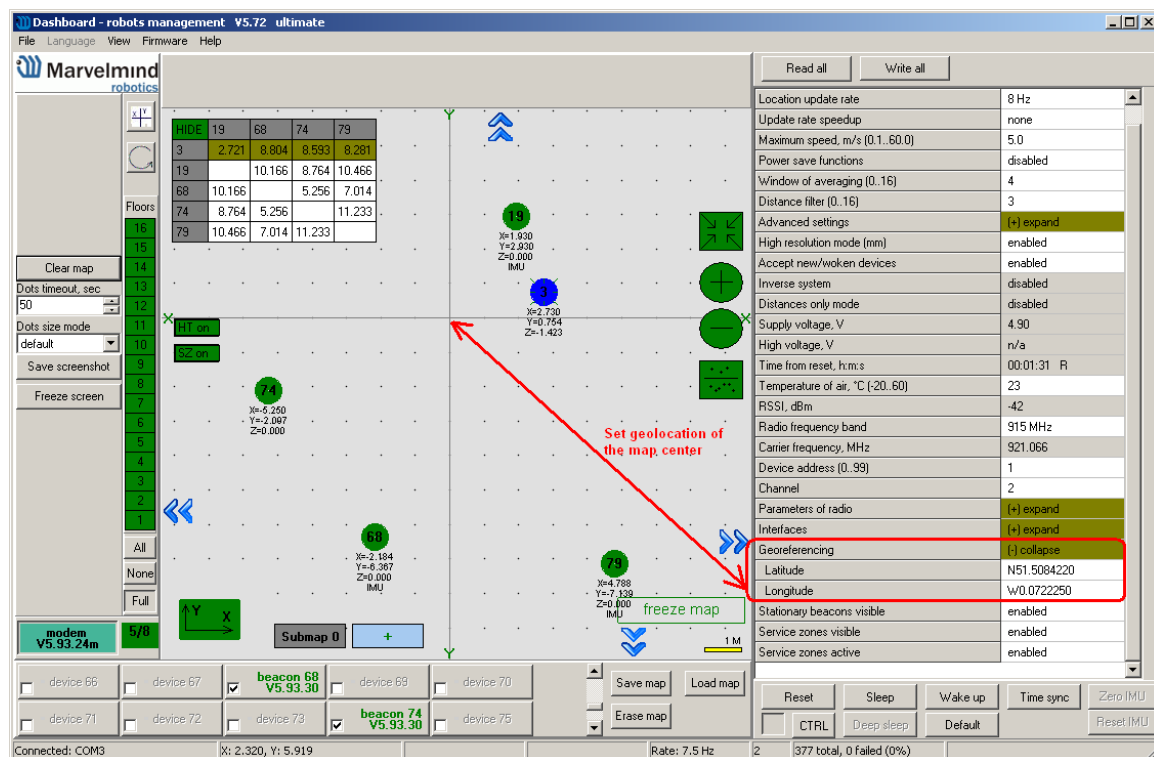
1. Connect to USB-host as an USB device of CDC class (virtual COM port in Windows, ttyUSB or ttyACM in Linux). In the Windows, it requires driver - the same driver as for modem. In Linux, the driver in most cases is not required, since the required driver is integrated into Linux kernel. Because real RS-232 is not used in the interface, parameters of serial port opened on the host (baudrate, number of bits, parity, etc) may be any.
2. Connect to UART on a hedgehog – 2 wires soldering to pins required. See the picture of beacon interface below. To have the location data out, it is sufficient to connect only 2 wires: GND and USART2\_TX. Logic level of UART transmitter is CMOS 3.3V. Default baudrate is 500 kbps, it is configurable from the Dashboard (see parameter “UART speed, bps” on above picture) from following list: 4.8, 9.6, 19.2, 38.4, 57.6, 115.2, 500 kbps. Format of data: 8 bit, no parity, 1 stop bit.

### 2.3.1 General agreements for coordinates translation

Marvelmind system measures position in form of rectangular Cartesian system coordinates (X, Y, Z), where Z in most cases is the height. For translation to GPS coordinates following agreements are used:

- Z axis is directed up, Z coordinate means altitude above sea level;
- Y axis is directed to north, so Y is latitude;
- X axis is directed to east, so X is longitude;
- point (X= 0, Y= 0) has GPS coordinates according to georeference point (by default: 0 ° North, 0 ° West);

Georeference coordinates can be set as shown on the screenshot:



GPS coordinates are calculated according to specified georeference point and WGS-84 Earth model.

More, detailed,

$$\text{Lat} = \text{Lat\_ref} + y * 9.013373$$

where

Lat - latitude, microdegrees

Lat\_ref - georeference latitude, microdegrees

y - y coordinates in Marvelmind system, meters

$$\text{Long} = \text{Long\_ref} + x * 8.98315 / \cos(\text{Lat\_ref} / 1000000)$$

Long - longitude, microdegrees

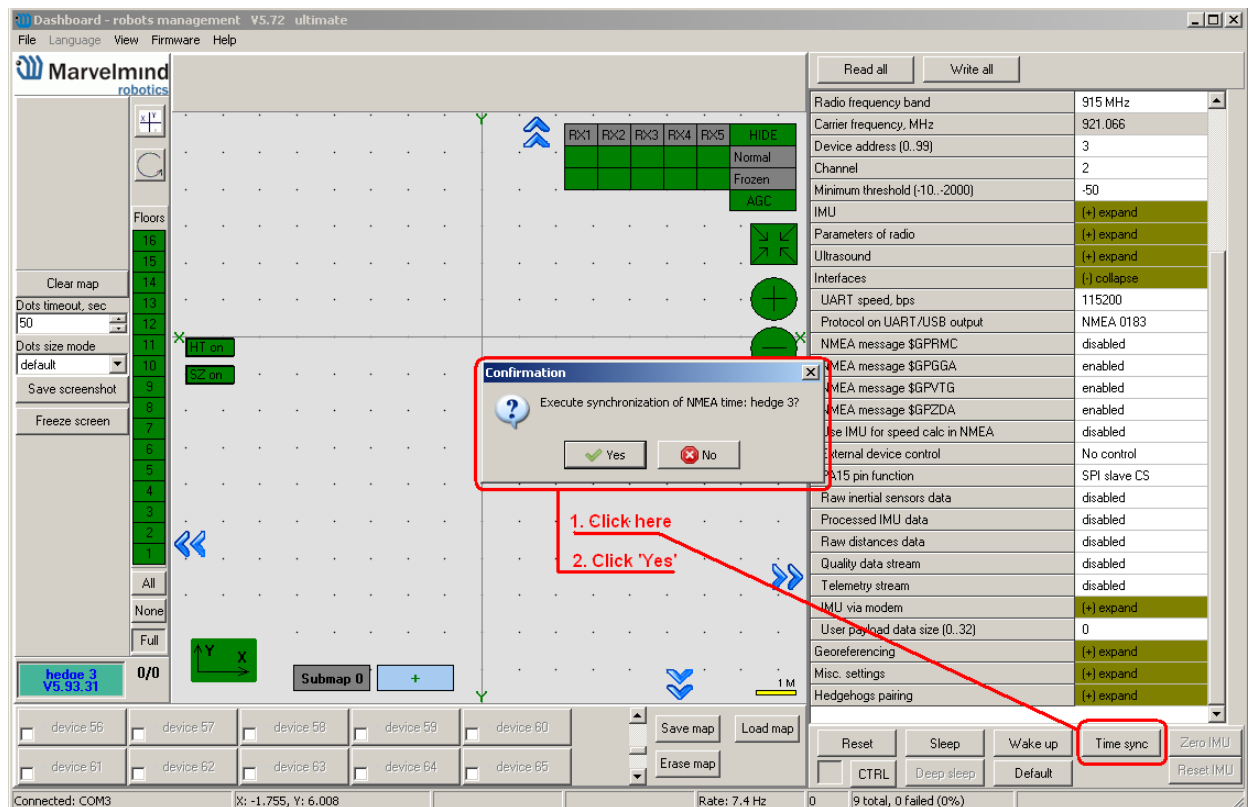
Long\_ref - georeference longitude, microdegrees

Lat\_ref - georeference latitude, microdegrees

x - x coordinates in Marvelmind system, meters

### 2.3.2 General agreements for time

After power on, mobile beacon counts time starting from 2016.08.01 00:00:00. User can synchronize time with computer clock as shown on following screenshot.



### 2.3.3 Description of “NMEA0183” messages implementation

NMEA 0183 messages are ASCII coded text frames, consist of several parts, separated by commas, and terminated by end of line. Before end of line, every message is finished by “\*” symbol, followed by two symbols of checksum, calculated according to NMEA 0183 standard. Each part of NMEA 0183 message represents certain parameter. Below is description of all supported messages and parameter fields. Messages format is taken from NMEA 0183 standard version 3.01, January 1, 2002.

#### 1. \$GPRMC -Recommended Minimum Specific GNSS Data

##### Supported hardware:

Super-Beacon:	supported
Industrial Super-Beacon:	supported
Modem HW5.1:	supported (starting from SW V7.000)
Super-Modem:	supported (starting from SW V7.000)
Mini-Rx (Badge, Helmet, etc.):	supported with <a href="#">the UART cable for Mini-Rx</a>
Mini-Tx:	not supported in the current HW version
Modem HW4.9:	not supported
Beacon HW4.9:	supported
Beacon HW4.5:	supported

General format from NMEA 0183 standard:

\$GPRMC, hhmmss.ss, A, llll.ll, a, yyyyy.yy, a.x.x,x.x,xxxxxx,x.x,a \*hh <CR> <LF>

The diagram illustrates the structure of the \$GPRMC message. It shows the following fields and their meanings:

- hhmmss.ss**: UTC of position fix
- A**: Status
  - A = Data valid
  - V = Navigation receiver warning
- llll.ll**: Latitude, N/S
- a**: Mode Indicator
- yyyyy.yy**: Date: ddmmyy
- a.x.x,x.x,xxxxxx,x.x,a**:
  - x.x**: Speed over ground, knots
  - xxxxxx**: Course Over Ground, degrees True
  - x.x**: Longitude, E/W
  - a**: Magnetic variation, degrees E/W

Description of fields implementation:

- 1.1. '\$GPRMC' – designation of message type
- 1.2. 'hhmmss.ss' – UTC position fix

According to general agreements, time is counted from default 2016.01.01 or synchronized with computer clock.

- 1.3. 'A' – status

'A' value is sent if last position update was successful

'V' value is sent if any error occurred in last position update

- 1.4. 'llll.ll, a' – latitude, N/S

According to general agreements (see above), latitude corresponds to the Y coordinate relative to georeference location. Latitude is presented with 6 digits of decimal-fraction of minutes, which gives resolution not more than 2 mm,

- 1.5. 'yyyyy.yyyyyy, a' – longitude, E/W

According to general agreements (see above), longitude corresponds to the X coordinate relative to georeference location. Longitude is presented with 6 digits of decimal-fraction of minutes, which gives resolution not more than 2 mm.

- 1.6. 'x.x' – speed over ground, knots

Marvelmind system measures the coordinates; the speed is calculated from dynamics of coordinates with applying of some filtering. Optionally, it can use IMU fusion for speed calculation.

- 1.7. 'xxxxxx' - date: ddmmyy

According to general agreements, time is counted from default 2016.01.01 or synchronized with computer clock.

### 1.8. 'x.x,a' - magnetic variation

This parameter value is always a null field.

### 1.9. 'a' - mode indicator

'A' value (autonomous mode) is sent if last position update was successful

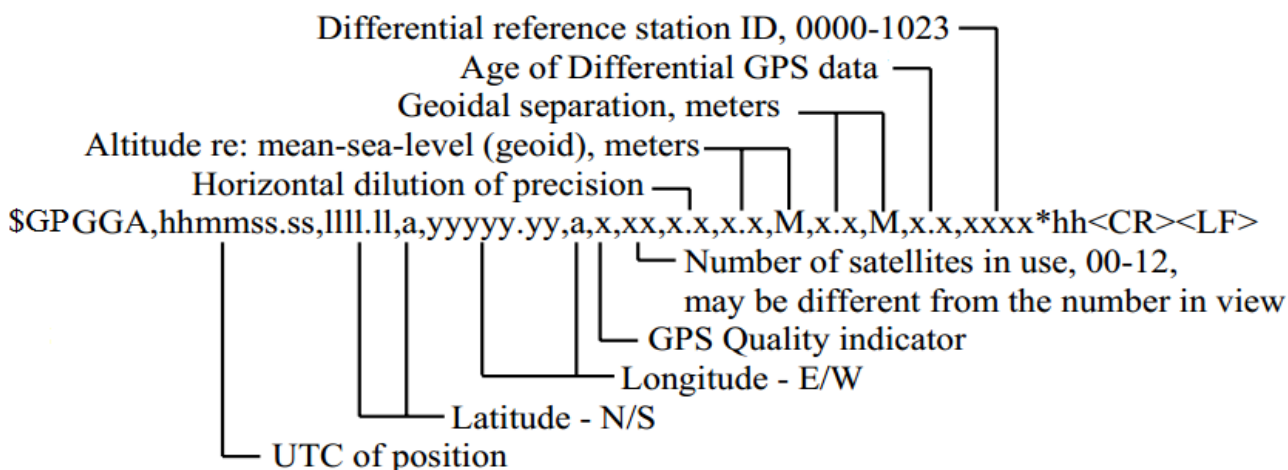
'N' value (data not valid) is sent if any error occurred in last position update

## 2. \$GPGGA -Global Positioning System Fix Data

### Supported hardware:

Super-Beacon:	supported
Industrial Super-Beacon:	supported
Modem HW5.1:	on demand
Super-Modem:	on demand
Mini-Rx (Badge, Helmet, etc.):	supported with <a href="#">the UART cable for Mini-Rx</a>
Mini-Tx:	not supported in the current HW version
Modem HW4.9:	not supported
Beacon HW4.9:	supported
Beacon HW4.5:	supported

General format from NMEA 0183 standard:



Description of fields implementation:

### 2.1. '\$GPGGA' – designation of message type

### 2.2. 'hhmmss.ss' – UTC position fix

According to general agreements, time is counted from default 2016.01.01 or synchronized with computer clock.

### 2.3. 'llll.ll, a' – latitude, N/S

According to general agreements (see above), latitude corresponds to the Y coordinate relative to georeference location. Latitude is presented with 6 digits of decimal-fraction of minutes, which gives resolution not more than 2 mm

### 2.4. 'yyyy.yy, a' – longitude, E/W

According to general agreements (see above), longitude corresponds to the X coordinate relative to georeference location. Longitude is presented with 6 digits of decimal-fraction of minutes, which gives resolution not more than 2 mm

### 2.5. 'x' – GPS quality indicator

'1' (GPS SPS Mode, fix valid) value is sent if last position update was successful

'0' (Fix not available or invalid) value is sent if any error occurred in last position update

### 2.6. 'xx' – number of satellites in use

Always '08' in current implementation.

### 2.7. 'x.x' – horizontal dilution of precision

Always '1.2' in current implementation.

### 2.8. 'x.x, M' – altitude re: mean-sea-level (geoid), meters

This corresponds to the Z coordinate according to general agreements.

### 2.9. 'x.x, M' – geoidal separation, meters

Always '0.0, M' value is transmitted.

### 2.10. 'x.x' – age of differential GPS data

This parameter value is always a null field, DGPS is not used.

### 2.11. 'xxxx' – differential reference station ID

This parameter value is always a null field.

## 3. \$GPVTG -Course Over Ground and Ground Speed

### Supported hardware:

Super-Beacon:	supported
Industrial Super-Beacon:	supported
Modem HW5.1:	on demand
Super-Modem:	on demand
Mini-Rx (Badge, Helmet, etc.):	supported with <a href="#">the UART cable for Mini-Rx</a>
Mini-Tx:	not supported in the current HW version
Modem HW4.9:	not supported
Beacon HW4.9:	supported
Beacon HW4.5:	supported

General format from NMEA 0183 standard:

**\$GPVTG,x.x,T,x.x,M,x.x,N,x.x,K,a\*hh<CR><LF>**

Mode Indicator  
Speed over ground, km/hr  
Speed over ground, knots  
Course over ground, degrees Magnetic  
Course over ground, degrees True

Description of fields implementation:

### 3.1. '\$GPVTG' – designation of message type

### 3.2 . 'x.x, T' – course over ground, degrees True

According to NMEA standard, the course is the angle between vector of speed and direction to the north. As shown in general agreements above, the Y axis is taken as direction to north.

### 3.3. 'x.x, M' – course over ground, degrees Magnetic

In current implementation, magnetic course is same as true course.

### 3.4. 'x.x, N' – speed over ground, knots

Marvelmind system measures the coordinates; the speed is calculated from dynamics of coordinates with applying of some filtering. Optionally, it can use IMU fusion for speed calculation.

### 3.5. 'x.x, K' – speed over ground, km/hr

It is the same speed in another units

### 3.6. 'a' – mode indicator

'A' value (autonomous mode) is sent if last position update was successful

'N' value (data not valid) is sent if any error occurred in last position update



## 4. \$GPZDA –Time & Date

### Supported hardware:

Super-Beacon:	supported
Industrial Super-Beacon:	supported
Modem HW5.1:	on demand
Super-Modem:	on demand
Mini-Rx (Badge, Helmet, etc.):	supported with <a href="#">the UART cable for Mini-Rx</a>
Mini-Tx:	not supported in the current HW version
Modem HW4.9:	not supported
Beacon HW4.9:	supported
Beacon HW4.5:	supported

General format from NMEA 0183 standard:

**\$GPZDA,hhmmss.ss,xx,xx,xxxx,xx,xx\*hh<CR><LF>**

Local zone minutes , 00 to +59  
Local zone hours , 00 to ±13 hrs  
Year  
Month, 01 to 12  
Day, 01 to 31  
UTC

According to general agreements, time is counted from default 2016.01.01 or synchronized with computer clock.

Description of fields implementation:

#### 4.1. '\$GPZDA' – designation of message type

#### 4.1. 'hhmmss.ss' – UTC

Time (hours, minutes, seconds).

#### 4.2. 'xx' – day, 01 to 31

Day.

#### 4.3. 'xx' – month, 01 to 12

Month.

#### 4.4. 'xxxx' – year

Year.

#### 4.4. 'xx – local zone hours

Local zone is always "00" hours.

#### 4.5. 'xx – local zone minutes

Local zone is always "00" minutes.

## 5. \$GPHDT – Heading

### Supported hardware:

Super-Beacon:	supported
Industrial Super-Beacon:	supported
Modem HW5.1:	on demand
Super-Modem:	on demand
Mini-Rx (Badge, Helmet, etc.):	supported with <a href="#">the UART cable for Mini-Rx</a>
Mini-Tx:	not supported
Modem HW4.9:	not supported
Beacon HW4.9:	not supported
Beacon HW4.5:	not supported

Note you need [MMSW0002](#) license to enable streaming of this packet.

General format from NMEA 0183 standard:

\$GPHDT,x.x,T\*hh<CR><LF>  
    └─┬─ Heading, degrees True

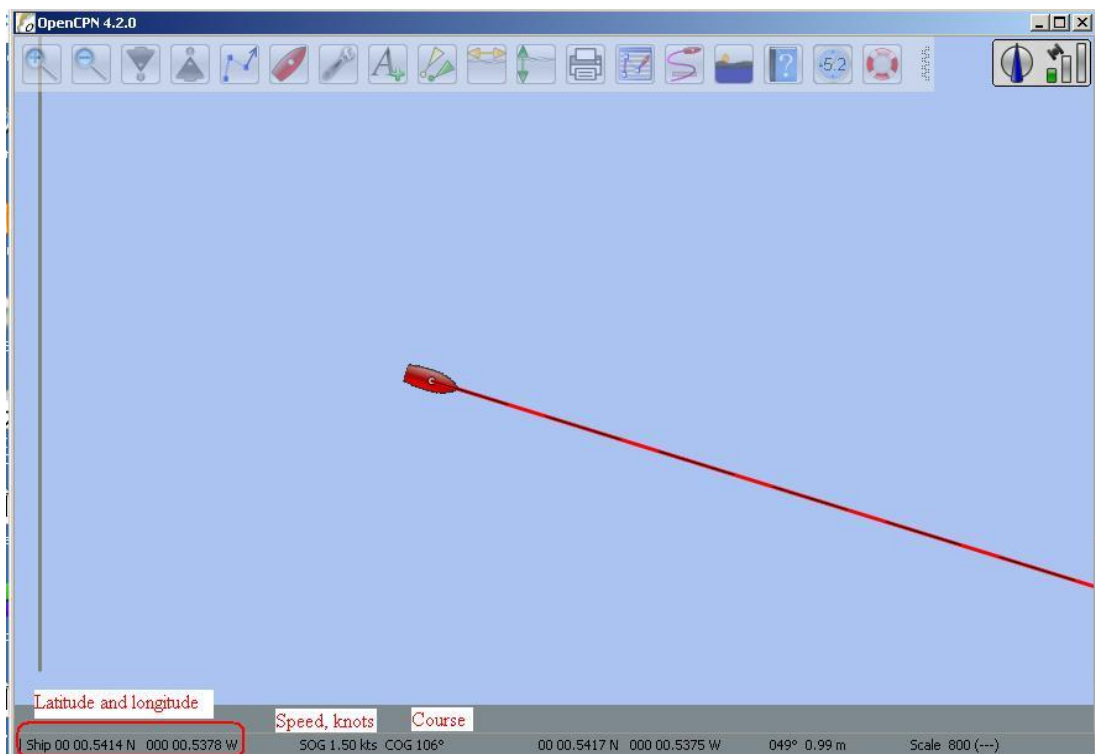
### 5.1. '\$GPHDT' – designation of message type

### 5.2. 'x.x, T' – heading, degrees True

This is a heading angle calculated by using paired beacons or paired microphones feature with fusion with the gyroscope.

### 2.3.4 Examples of NMEA data receiving

On the next screenshot is example of data, received from mobile beacon, connected via USB (virtual COM port) to the OpenCPN software, running on the computer under MS Windows.



## 3. Protocols of communication via USB (virtual UART)

### 3.1 'Marvelmind' protocol for streaming

All packets described in [corresponding section](#) for UART are also available via USB (virtual UART).

Note these data are also available for mini-Tx and for mini-Rx without 'UART Cable for Mini-Rx'.

Streaming is terminated for 5 seconds if Marvelmind device receives any request according to [this protocol](#).

## 3.2 Protocol of reading/writing data from/to user device

All packets described in [corresponding section](#) for UART are also available via USB (virtual UART).

Note these data are also available for mini-Tx and for mini-Rx without 'UART Cable for Mini-Rx'.

### 3.3 NMEA0183 communication protocol

All packets described in [corresponding section](#) for UART are also available via USB (virtual UART).  
Note these data are also available for mini-Tx and for mini-Rx without 'UART Cable for Mini-Rx'.  
Streaming is terminated for 5 seconds if Marvelmind device receives any request according to [this protocol](#).

### 3.4 Protocol of data exchange with modem via USB interface

This protocol is used by Dashboard software and Marvelmind API described in next chapter.

Modem connects to USB-host as USB device of CDC class (virtual COM port in Windows, ttyUSB or ttyACM in Linux).

Because real RS-232 is not used in this interface, parameters of serial port set on the host (baudrate, number of bits, parity, etc.) may be any

Data is in binary format

«Network address» of device connected via USB is **0xff**

Multibyte numbers are transmitted starting from low byte (little endian format)

### 3.4.1 Reading the latest coordinates pack (firmware V5.13+)

#### Supported hardware:

Super-Beacon:	not supported
Industrial Super-Beacon:	not supported
Modem HW5.1:	supported
Super-Modem:	supported
Mini-Rx (Badge, Helmet, etc.):	not supported
Mini-Tx:	not supported
Modem HW4.9:	supported
Beacon HW4.9:	not supported
Beacon HW4.5:	not supported

#### Format of request frame (from host to modem)

Offset	Size (bytes)	Type	Description	Value
0	1	uint8_t	Address of modem	0xff
1	1	uint8_t	Type of packet	0x03
2	2	uint16_t	Code of data in packet	0x4110
4	2	uint16_t	Access mode	0x0000
6	2	uint16_t	CRC-16 (see appendix 1)	0xc004

#### Format of answer frame (from modem to host)

Offset	Size (bytes)	Type	Description	Value
0	1	uint8_t	Address of modem	0xff
1	1	uint8_t	Type of packet	0x03
2	1	uint8_t	Number of bytes of data transmitting	0x64
3	100 (0x64)	100 bytes	Data structure (see lower)	
103	2	uint16_t	CRC-16 (see appendix 1)	

Format of error reply is described in Appendix 2.

#### Format of data field (100 bytes)

Offset	Size (bytes)	Description
0	96 (6*16)	Six last coordinates structures received by modem (see lower)
96	1	Byte of flags: Bit 0...1: reserved Bit 2: 1 = user data available (see section 12) Bit 3...7: reserved
97	3	Reserved

#### Format of coordinates structure (16 bytes)

Offset	Size (bytes)	Description
0	1	Address of device
1	4	Coordinate X, mm (int32_t)
5	4	Coordinate Y, mm (int32_t)
9	4	Coordinate Z, mm (int32_t)
13	1	Byte of flags: Bit 0: 1 – no relevant coordinates (red mode in dashboard) Bit 1: 1 – temporary mobile beacon on frozen map (blue mode) Bit 2: 1 – beacon is used for hedgehog positioning
14	2	Reserved (0)



### 3.4.2 Reading/writing modem configuration

#### 3.4.2.1 Reading modem configuration (firmware V5.30+)

##### Supported hardware:

Super-Beacon:	not supported
Industrial Super-Beacon:	not supported
Modem HW5.1:	supported
Super-Modem:	supported
Mini-Rx (Badge, Helmet, etc.):	not supported
Mini-Tx:	not supported
Modem HW4.9:	supported
Beacon HW4.9:	not supported
Beacon HW4.5:	not supported

##### Format of request frame (from host to modem)

Offset	Size (bytes)	Type	Description	Value
0	1	uint8_t	Address of modem	0xff
1	1	uint8_t	Type of packet	0x03
2	2	uint16_t	Code of data in packet	0x5000
4	2	uint16_t	Access mode	0x0000
6	2	uint16_t	CRC-16 (see appendix 1)	0x0550

##### Format of answer frame (from modem to host)

Offset	Size (bytes)	Type	Description	Value
0	1	uint8_t	Address of modem	0xff
1	1	uint8_t	Type of packet	0x03
2	1	uint8_t	Number of bytes of data transmission	0x30
3	0x30	structure	Data structure (see below)	
0x33	2	uint16_t	CRC-16 (see appendix 1)	

Format of error reply is described in Appendix 2.

### 3.4.2.2 Writing modem configuration

**Warning!** To write modem configuration you must read configuration, setup the data fields described in following section, and then write it. Do not change any other bytes in structure, this may degrade the work of modem

#### Format of request frame (from host to modem)

Offset	Size (bytes)	Type	Description	Value
0	1	uint8_t	Address of modem	0xff
1	1	uint8_t	Type of packet	0x10
2	2	uint16_t	Code of data in packet	0x5000
4	2	uint16_t	Access mode	0x0000
6	1	uint8_t	Number of bytes of data transmission	0x30
7	0x30	structure	Data structure (see below)	
0x37	2	uint16_t	CRC-16 (see appendix 1)	

#### Format of answer frame (from modem to host)

Offset	Size (bytes)	Type	Description	Value
0	1	uint8_t	Address of modem	0xff
1	1	uint8_t	Type of packet	0x10
2	2	uint16_t	Code of data	0x5000
4	2	uint16_t	reserved	
6	2	uint16_t	CRC-16 (see appendix 1)	

Format of error reply is described in Appendix 2.

### 3.4.2.3 Structure of modem configuration data

Many fields of data structure are not explained. Do not change the fields! They are used for adjustment system from the Dashboard program; unauthorized changing may degrade the work of modem

Offset	Size (bytes)	Type	Description
0	20	20 bytes	Not explained
20	1	int8_t	Temperature of air setting Vt (signed). Temperature is (Vt+23) °C
21	1	uint8_t	Address of the beacon that should have map coordinates X=0, Y=0
22	4	4 bytes	Not explained
26	1	uint8_t	Address of the beacon that should have map coordinates X>0,Y=0
27	1	uint8_t	Address of the beacon that should have map coordinates with Y>0
28	1	uint8_t	Control flags: Bit 0: not explained Bit 1: 1 - enabled filtering of mobile beacons movement Bit 2: not explained Bit 3: 1 - high resolution mode (output coordinates in mm instead cm) Bit 4: not explained Bit 5: 1 = mirroring of all map Bit 6: 1= power save mode (power save works only when all of the submaps are frozen) Bit 7: not explained
29	2	2 bytes	Not explained
31	1	uint8_t	N, determines maximum frequency of retrieving hedgehog coordinates $F(N) = 2^{(N-1)} \text{ Hz}$ , $N = 0 \dots 4$ , $F(5) = 12 \text{ Hz}$ , $F(6) = 16 \text{ Hz}$ , $F(7) = 16+ \text{ (maximum)}$
32	16	16 bytes	Not explained

### 3.4.3 Reading/writing submap configuration

#### Supported hardware:

Super-Beacon:	not supported
Industrial Super-Beacon:	not supported
Modem HW5.1:	supported
Super-Modem:	supported
Mini-Rx (Badge, Helmet, etc.):	not supported
Mini-Tx:	not supported
Modem HW4.9:	supported
Beacon HW4.9:	not supported
Beacon HW4.5:	not supported

#### 3.4.3.1 Reading submap configuration (firmware V5.30+)

##### Format of request frame (from host to modem)

Offset	Size (bytes)	Type	Description	Value
0	1	uint8_t	Address of modem	0xff
1	1	uint8_t	Type of packet	0x03
2	2	uint16_t	Code of data in packet	0x60XX where XX is number of submap
4	2	uint16_t	Access mode	0x0000
6	2	uint16_t	CRC-16 (see appendix 1)	

##### Format of answer frame (from modem to host)

Offset	Size (bytes)	Type	Description	Value
0	1	uint8_t	Address of modem	0xff
1	1	uint8_t	Type of packet	0x03
2	1	uint8_t	Number of bytes of data transmission	0x50 (80)
3	80	structure	Data structure (see below)	
83	2	uint16_t	CRC-16 (see appendix 1)	

Format of error reply is described in Appendix 2.

### 3.4.3.2 Writing submap configuration (firmware V5.30+)

Warning! To write submap configuration you must read configuration, setup the data fields described in following section, and then write it. Do not change any other bytes in structure, this may degrade the work of modem

#### Format of request frame (from host to modem)

Offset	Size (bytes)	Type	Description	Value
0	1	uint8_t	Address of modem	0xff
1	1	uint8_t	Type of packet	0x10
2	2	uint16_t	Code of data in packet	0x60XX where XX is number of submap
4	2	uint16_t	Access mode	0x0000
6	1	uint8_t	Number of bytes of data transmission	0x50 (80)
7	80	structure	Data structure (see below)	
87	2	uint16_t	CRC-16 (see appendix 1)	

#### Format of answer frame (from modem to host)

Offset	Size (bytes)	Type	Description	Value
0	1	uint8_t	Address of modem	0xff
1	1	uint8_t	Type of packet	0x10
2	2	uint16_t	Code of data	0x5000
4	2	uint16_t	reserved	
6	2	uint16_t	CRC-16 (see appendix 1)	

Format of error reply is described in Appendix 2.

### 3.4.3.3 Structure of submap configuration data

Many fields of data structure are not explained. Do not change the fields! They are used for adjustment system from the Dashboard program; unauthorized changing may degrade the work of modem

Offset	Size (bytes)	Type	Description
0	1	uint8_t	Address of starting beacon for building submap
1	1	uint8_t	Control word: Bit 0: 1 - submap is frozen (freeze submap) Bit 1: 1 - beacons are higher than hedgehogs Bit 2...4: not explained Bit 5: 1 - mirroring submap Bit 6...7: not explained
2	1	uint8_t	Limitation of distances: Bit 0...6: manual limitation distances (if bit 7 = 1) Bit 7: 0 - automatic limitation, 1 = manual
3	13	13 bytes	Not explained
16	2	int16_t	X shift of submap, cm
18	2	int16_t	Y shift of submap, cm
20	2	uint16_t	Rotation of submap, centidegrees
22	58	58 bytes	Not explained

### 3.4.4 Sleeping/waking up devices

#### Supported hardware:

Super-Beacon:	not supported
Industrial Super-Beacon:	not supported
Modem HW5.1:	supported
Super-Modem:	supported
Mini-Rx (Badge, Helmet, etc.):	not supported
Mini-Tx:	not supported
Modem HW4.9:	supported
Beacon HW4.9:	not supported
Beacon HW4.5:	not supported

#### Format of request frame (from host to modem)

Offset	Size (bytes)	Type	Description	Value
0	1	uint8_t	Address of device	0x01...0xfe
1	1	uint8_t	Type of packet	0x10
2	2	uint16_t	Code of data in packet	0xb006
4	2	uint16_t	Access mode	For wake: 0x0002 Others: 0x0001
6	1	uint8_t	Number of bytes of data transmission	0x08
7	1	uint8_t	Password, byte 0	0x2d
8	1	uint8_t	Password, byte 1	0x94
9	1	uint8_t	Password, byte 2	0x5e
10	1	uint8_t	Password, byte 3	0x81
11	1	uint8_t	Command: 0 – standard sleep 1 – deep sleep (wake only on HW reset) 2 – wake up from standard sleep 3...255 - reserved	0...2
12	3	3 bytes	reserved	
15	2	uint16_t	CRC-16 (see appendix 1)	

#### Format of answer frame for waking command (from modem to host)

Offset	Size (bytes)	Type	Description	Value
0	1	uint8_t	Address of device	0x01...0xfe
1	1	uint8_t	Type of packet	0x10
2	2	uint16_t	Code of data	0xb006
4	2	uint16_t	reserved	
6	2	uint16_t	CRC-16 (see appendix)	

#### Format of answer frame for sleeping commands (from modem to host)

Offset	Size (bytes)	Type	Description	Value
0	1	uint8_t	Address of modem	0xff
1	1	uint8_t	Type of packet (modem reply)	0x7f
2	2	uint16_t	Code of data	0xb006
4	2	uint16_t	reserved	
6	2	uint16_t	CRC-16 of bytes 0...5 (see appendix 1)	

8	1	uint8_t	Address of device	0x01...0xfe
9	1	uint8_t	Type of packet	0x10
10	2	uint16_t	Code of data	0xb006
12	2	uint16_t	reserved	
14	2	uint16_t	CRC-16 for bytes 8...13(see appendix)	

Format of error reply is described in Appendix 2.



### 3.4.5 Setting address of device

#### Supported hardware:

Super-Beacon:	not supported
Industrial Super-Beacon:	not supported
Modem HW5.1:	supported
Super-Modem:	supported
Mini-Rx (Badge, Helmet, etc.):	not supported
Mini-Tx:	not supported
Modem HW4.9:	supported
Beacon HW4.9:	not supported
Beacon HW4.5:	not supported

#### Format of request frame (from host to modem)

Offset	Size (bytes)	Type	Description	Value
0	1	uint8_t	Address of device	0x01...0xfe
1	1	uint8_t	Type of packet	0x10
2	2	uint16_t	Code of data in packet	0x0101
4	2	uint16_t	Access mode	0x0000
6	1	uint8_t	Number of bytes of data transmission	0x02
7	1	uint8_t	Code of data item (address)	0x00
8	1	uint8_t	New address of device	
9	2	uint16_t	CRC-16 (see appendix 1)	

### 3.4.6 Reading measured raw distances

#### Supported hardware:

Super-Beacon:	not supported
Industrial Super-Beacon:	not supported
Modem HW5.1:	supported
Super-Modem:	supported
Mini-Rx (Badge, Helmet, etc.):	not supported
Mini-Tx:	not supported
Modem HW4.9:	supported
Beacon HW4.9:	not supported
Beacon HW4.5:	not supported

This command is accessible in two modes:

- With code of data 0x4000 – reading last eight distances. Answer frame contains last 8 measured distances from the moment of request
- With code of data 0x4001 – reading all distances frame by frame. Answer frame for every next request contains next 8 saved measured distances. When all table of distances is transmitted, it starts from the beginning

#### Format of request frame (from host to modem)

Offset	Size (bytes)	Type	Description	Value
0	1	uint8_t	Address of modem	0xff
1	1	uint8_t	Type of packet	0x03
2	2	uint16_t	Code of data in packet	0x4000 or 0x4001
4	2	uint16_t	Access mode	0x0000
6	2	uint16_t	CRC-16 (see appendix 1)	

#### Format of answer frame (from modem to host)

Offset	Size (bytes)	Type	Description	Value
0	1	uint8_t	Address of modem	0xff
1	1	uint8_t	Type of packet	0x03
2	1	uint8_t	Number of bytes of data transmitting	0x28
3	40 (0x28)	40 bytes	Data structure (see lower)	
43	2	uint16_t	CRC-16 (see appendix 1)	

Format of error reply is described in Appendix 2.

#### Format of data field (40 bytes)

Offset	Size (bytes)	Description
0	32 (8*4)	Eight raw distances structures (see lower)
32	8	Reserved

#### Format of distance structure (4 bytes)

Offset	Size (bytes)	Description
0	1	Address of ultrasonic receiver
1	1	Address of ultrasonic transmitter
2	2	Measured distance between devices, mm (uint16_t)

### 3.4.7 Reading beacons' state (firmware V5.33+)

#### Supported hardware:

Super-Beacon:	not supported
Industrial Super-Beacon:	not supported
Modem HW5.1:	supported
Super-Modem:	supported
Mini-Rx (Badge, Helmet, etc.):	not supported
Mini-Tx:	not supported
Modem HW4.9:	supported
Beacon HW4.9:	not supported
Beacon HW4.5:	not supported

#### Format of request frame (from host to modem)

Offset	Size (bytes)	Type	Description	Value
0	1	uint8_t	Address of device	0x01...0xfe
1	1	uint8_t	Type of packet	0x03
2	2	uint16_t	Code of data in packet	0x0003
4	2	uint16_t	Access mode	0x0002
6	2	uint16_t	CRC-16 (see appendix 1)	

#### Format of answer frame (from modem to host)

Offset	Size (bytes)	Type	Description	Value
0	1	uint8_t	Address of device	0x01...0xfe
1	1	uint8_t	Type of packet	0x03
2	1	uint8_t	Number of bytes of data transmission	0x20
3	32	32 bytes	Data structure (see lower)	
35	2	uint16_t	CRC-16 (see appendix 1)	

Format of error reply is described in Appendix 2.

#### Format of data field:

Offset	Size (bytes)	Type	Description
0	4	uint32_t	Time of work from reset or wake-up (seconds)
4	1	uint8_t	R, radio RSSI register value (received signal strength indicator). If $R > 128$ , $RSSI \text{ (dBm)} = (R - 256) / 2 - 74$ If $R \leq 128$ , $RSSI \text{ (dBm)} = (R / 2) - 74$
5	1	uint8_t	Not explained
6	1	int8_t	Measured temperature $V_t$ (signed). Temperature is $(V_t + 23) \text{ } ^\circ\text{C}$
7	2	uint16_t	Bit 0...11: power supply voltage, mV Bit 12...13: not explained Bit 14: 1: low power, device will enter sleep after short time Bit 15: 1: very low power, device will enter deep sleep after short time
9	23	23 bytes	Not explained

### 3.4.8 Marvelmind robots control commands

#### Supported hardware:

Super-Beacon:	not supported
Industrial Super-Beacon:	not supported
Modem HW5.1:	supported
Super-Modem:	supported
Mini-Rx (Badge, Helmet, etc.):	not supported
Mini-Tx:	not supported
Modem HW4.9:	supported
Beacon HW4.9:	not supported
Beacon HW4.5:	not supported

#### 3.4.8.1 Robot control command

##### Format of request frame (from host to modem)

Offset	Size (bytes)	Type	Description	Value
0	1	uint8_t	Address of robot	0x01...0xfe
1	1	uint8_t	Type of packet	0x10
2	2	uint16_t	Code of data in packet	0x1000
4	2	uint16_t	Access mode	0x0001
6	1	uint8_t	Number of bytes of data transmission	0x10
7	16 (0x10) bytes	uint8_t	Robot control data (see lower)	
23	2	uint16_t	CRC-16 (see appendix 1)	

##### Format of answer frame (from modem to host)

Offset	Size (bytes)	Type	Description	Value
0	1	uint8_t	Address of modem	0xff
1	1	uint8_t	Type of packet (modem reply)	0x7f
2	2	uint16_t	Code of data	0x1000
4	2	uint16_t	reserved	
6	2	uint16_t	CRC-16 of bytes 0...5(see appendix 1)	
8	1	uint8_t	Address of robot	0x01...0xfe
9	1	uint8_t	Type of packet (robot reply)	0x10
10	2	uint16_t	Code of data	0x1000
12	2	uint16_t	reserved	
14	2	uint16_t	CRC-16 of bytes 8...13(see appendix 1)	

Format of error reply is described in Appendix 2.

##### Format of robot control data:

Offset	Size (bytes)	Type	Description
0	1	uint8_t	Mode of control: 0 - no control (wait mode) 1 - motors power control 2 - speed control 3 - writing movement program 4 - pause movement program 5 - continue movement after pause
1	1	uint8_t	Code of operation: 0 - move forward 1 - move backward 2 - rotate clockwise

			3 - rotate counterclockwise 4 - pause for given time (for mode 3) 5 - repeat movement program from start (for mode 3) 6 - move to given point by coordinates (for mode 3) 7 - setup movement speed (for mode 3)
2	1	uint8_t	Control byte 1: For mode 1: power on motors, % For mode 2: speed of movement, % For mode 3: number of the program step (starting from zero)
3	2	Int16_t	Data for program (mode 3): Code of operation 0 or 1: distance of movement, cm Code of operation 2 or 3: angle of rotation, degrees Code of operation 4: time of pause, ms Code of operation 6: X coordinate of movement target, cm Code of operation 7: speed of movement, %
5	1	uint8_t	For mode 3: total number of steps in program.
6	2	int16_t	Additional data for program (mode 3): Code of operation 6: Y coordinate of movement target, cm
8	2	int16_t	Code of operation 6: Z coordinate of movement target, cm
10	6	6 bytes	Reserved (0)

Some comments for this complicated command.

There are three main modes of robot control specified in byte 0 of robot control structure:

- power control (mode 1)
- speed control (mode 2)
- move by program (mode 3)

Mode 1 and mode 2 are generally used for test purposes. In mode 1 robot moves forward, backward, rotates left or right with selected power on motors. In mode 2 robot makes the same but adjusting power to keep selected speed. The power or speed is set in byte 2 of structure, type of movement - in byte 1.

Mode 4 and mode 5 are special commands for pausing movement during program execution and continuing movement after pause.

The main mode for moving on complex trajectories is mode 3.

It allows to program to robot the sequence of primitive actions, which combination builds the trajectory. Each item of the sequence should be sent by one command of this type. Each command should contain the number of the current step in the byte 2 of robot control structure, and total number of steps in the byte 5.

In the byte 1 of robot control structure the type of primitive movement is specified. Parameters of the primitive movement are specified in fields "data for program" (bytes 3...4) and "additional data for program" (bytes 6...7).

So, the following primitives are available:

- move forward by given distance;
- move backward by given distance;
- rotate clockwise by given angle;
- rotate counterclockwise by given angle;
- pause by given time;
- restart the movement program from first item (for looping movements);
- move to given point (X, Y) in Marvelmind navigation system coordinates;
- change movement speed.

Robot begins execution of the program after receiving the sequence of primitives. After program execution, robot stops. But if the program contains item with code of operation

5 (repeat from start), the program repeats loop which will be executed forever, until receiving stop command or uploading new program.

### 3.4.8.2 Stop robot

#### Format of request frame (from host to modem)

Offset	Size (bytes)	Type	Description	Value
0	1	uint8_t	Address of robot	0x01...0xfe
1	1	uint8_t	Type of packet	0x10
2	2	uint16_t	Code of data in packet	0x403
4	2	uint16_t	Access mode	0x0001
6	1	uint8_t	Number of bytes of data transmission	0x04
7	4 bytes	4 bytes	Reserved (0)	0
11	2	uint16_t	CRC-16 (see appendix 1)	

#### Format of answer frame (from modem to host)

Offset	Size (bytes)	Type	Description	Value
0	1	uint8_t	Address of modem	0xff
1	1	uint8_t	Type of packet (modem reply)	0x7f
2	2	uint16_t	Code of data	0x403
4	2	uint16_t	reserved	
6	2	uint16_t	CRC-16 of bytes 0...5 (see appendix 1)	
8	1	uint8_t	Address of robot	0x01...0xfe
9	1	uint8_t	Type of packet (robot reply)	0x10
10	2	uint16_t	Code of data	0x403
12	2	uint16_t	reserved	
14	2	uint16_t	CRC-16 of bytes 8...13 (see appendix 1)	

Format of error reply is described in Appendix 2.

This command simply terminates execution of any robot movement or program of movements. The robot stops and waits for new commands.

### 3.4.9 Reading/writing device control settings (firmware V6.01+)

#### Supported hardware:

Super-Beacon:	not supported
Industrial Super-Beacon:	not supported
Modem HW5.1:	supported
Super-Modem:	supported
Mini-Rx (Badge, Helmet, etc.):	not supported
Mini-Tx:	not supported
Modem HW4.9:	supported
Beacon HW4.9:	not supported
Beacon HW4.5:	not supported

Note size of data for these frames is 8 bytes for DSP beacon and 16 bytes for HW4.9 beacon.

#### 3.4.9.1 Reading device control settings

##### Format of request frame (from host to modem)

Offset	Size (bytes)	Type	Description	Value
0	1	uint8_t	Address of device (beacon/modem)	0x01...0xfe or 0xff
1	1	uint8_t	Type of packet	0x03
2	2	uint16_t	Code of data in packet	0x1201
4	2	uint16_t	Access mode	0x0001
6	2	uint16_t	CRC-16 (see appendix 1)	

##### Format of answer frame (from modem to host)

Offset	Size (bytes)	Type	Description	Value
0	1	uint8_t	Address of modem	0xff
1	1	uint8_t	Type of packet	0x03 if request was sent to modem 0x7f if request was sent to beacon
2	1	uint8_t	Number of bytes of data transmission	0x08/0x10
3	8	structure	Data structure (see section 9.3). Relevant only if request was sent to modem (0xff)	
11	2	uint16_t	CRC-16 of bytes 0...10 (see appendix 1)	
following data will be received if the request was sent to beacon				
13	1	uint8_t	Address of device	0x01...0xfe
14	1	uint8_t	Type of packet	0x03
15	1	uint8_t	Number of bytes of data transmission	0x08/0x10
16	8	structure	Data structure (see below).	
24	2	uint16_t	CRC-16 of bytes 12...22 (see appendix 1)	

Format of error reply is described in Appendix 2.



### 3.4.9.2 Write device control settings

#### Format of request frame (from host to modem)

Offset	Size (bytes)	Type	Description	Value
0	1	uint8_t	Address of device (beacon/modem)	0x01...0xfe or 0xff
1	1	uint8_t	Type of packet	0x10
2	2	uint16_t	Code of data in packet	0x1201
4	2	uint16_t	Access mode	0x0001
6	1	uint8_t	Number of bytes of data transmission	0x08/0x10
7	8	structure	Data structure (see below)	
15	2	uint16_t	CRC-16 (see appendix 1)	

#### Format of answer frame (from modem to host)

Offset	Size (bytes)	Type	Description	Value
0	1	uint8_t	Address of modem	0xff
1	1	uint8_t	Type of packet	0x03 if request was sent to modem 0x7f if request was sent to beacon
2	2	uint16_t	Code of data	0x1201
4	2	uint16_t	reserved	
6	2	uint16_t	CRC-16 of bytes 0...5 (see appendix 1)	
following data will be received of the request was sent to beacon (with address 0x01...0xfe)				
8	1	uint8_t	Address of device	0x01...0xfe
9	1	uint8_t	Type of packet	0x10
10	2	uint16_t	Code of data	0x1201
12	2	uint16_t	reserved	
14	2	uint16_t	CRC-16 of bytes 8...13 (see appendix 1)	

Format of error reply is described in Appendix 2.

### 3.4.9.3 Format of control settings payload data

Offset	Size (bytes)	Type	Description
0	1	uint8_t	Flags: Bit 0...5: not explained, should be always zero! Bit 6: 0 - stationary beacon mode, 1 - hedgehog mode Bit 7: reserved (0)
1	1	uint8_t	UART baudrate setting: 0: 500000 bps (default value) 1: 4800 bps 2: 9600 bps 3: 19200 bps 4: 38400 bps 5: 57600 bps 6: 115200 bps 7...255: reserved
2	1	uint8_t	Reserved (0)
3	1	uint8_t	Bit 0...3: radio profile: 0: 38.4 kbps 1: 150 kbps 2: 500 kbps 3...7: reserved Bit 4...6: radio band: 0: 433 MHz 1: 868 MHz 2: 915 MHz 3: 315 MHz 4...7: reserved Bit 7: reserved
4	1	uint8_t	Type of UART/USB output: 0: Marvelmind protocol 1: NMEA0183
5	1	uint8_t	Mask of NMEA frames to send in NMEA0183 mode: Bit 0: 1 - send \$GPRMC frame Bit 1: 1 - send \$GPGGA frame Bit 2: 1 - send \$GPVTG frame Bit 3: 1 - send \$GPZDA frame Bit 4...7: reserved (0)
6	1	uint8_t	Number of bytes of user payload data for sending from this hedgehog to modem (0...32)
7	1	uint8_t	Mask of IMU data for sending to modem in 'IMU via modem' mode: Bit 0: IMU fusion location Bit 1: quaternion Bit 2: speed Bit 3: acceleration Bit 4: raw accelerometer Bit 5: raw gyro Bit 6: raw compass Bit 7: 0 = send IMU fusion, 1 = send raw IMU
Next fields are available for HW4.9 beacon only			

8	1	uint8_t	Bit 0...6: interval of streaming telemetry (0 = no stream) Bit 7: reserved (0)
9	1	uint8_t	Bit 0: use IMU for speed calculation Bit 1...7 – reserved (0)
10	6	6 bytes	Reserved (0)

**Warning!** If you change radio profile on beacon connected by radio, the radio connection will be lost. If you need to switch the profile, switch the radio profile for all beacons one after another, and then switch radio profile for modem. All beacons should be available on new radio profile after few seconds.

### 3.4.10 Reading list of devices in network (firmware V6.01+)

#### Supported hardware:

Super-Beacon:	not supported
Industrial Super-Beacon:	not supported
Modem HW5.1:	supported
Super-Modem:	supported
Mini-Rx (Badge, Helmet, etc.):	not supported
Mini-Tx:	not supported
Modem HW4.9:	supported
Beacon HW4.9:	not supported
Beacon HW4.5:	not supported

#### Format of request frame (from host to modem)

Offset	Size (bytes)	Type	Description	Value
0	1	uint8_t	Address of modem	0xff
1	1	uint8_t	Type of packet	0x03
2	2	uint16_t	Code of data in packet	0x31xx where xx is number of devices group
4	2	uint16_t	Access mode	0x0000
6	2	uint16_t	CRC-16 (see appendix 1)	

#### Format of answer frame (from modem to host)

Offset	Size (bytes)	Type	Description	Value
0	1	uint8_t	Address of modem	0xff
1	1	uint8_t	Type of packet	0x03
2	1	uint8_t	Number of bytes of data transmitting	0x72
3	1	uint8_t	Total number of devices in network (K)	
4	112	112 bytes	0...16 structures of information about device in network, see description lower	
116	1	uint8_t	Reserved	0x00
117	2	uint16_t	CRC-16 (see appendix 1)	

Format of error reply is described in Appendix 2.

#### Format of data about device in network (7 bytes)

Offset	Size (bytes)	Description
0	1	Address of device (0x01...0xfe)
1	1	Major version of firmware
2	1	Minor version of firmware
3	1	Bit 0...5: Type of device:

		10: Wheel robot 12: Crawler robot 22: Beacon HW V4.5 23: Beacon HW V4.5 (hedgehog mode) 24: Modem (HW V4.5/4.9) 30: Beacon HW V4.9 31: Beacon HW V4.9 (hedgehog mode) 32: Mini-RX beacon 36: Mini TX beacon (HW V5.07) 37: Industrial-TX beacon 41: Industrial-RX beacon 42: Super-Beacon 43: Super-Beacon (hedgehog mode) 44: Industrial Super-Beacon 45: Industrial Super-Beacon (hedgehog mode) Bit 6: 1 -more than one device with this address exist Bit 7: 1 - sleeping mode
4	1	Second minor version of firmware

### 3.4.11 Reading version of firmware

#### Supported hardware:

Super-Beacon:	supported
Industrial Super-Beacon:	supported
Modem HW5.1:	supported
Super-Modem:	supported
Mini-Rx (Badge, Helmet, etc.):	supported
Mini-Tx:	supported
Modem HW4.9:	supported
Beacon HW4.9:	supported
Beacon HW4.5:	supported

#### Format of request frame (from host to modem)

Offset	Size (bytes)	Type	Description	Value
0	1	uint8_t	Address of device	0xff
1	1	uint8_t	Type of packet	0x03
2	2	uint16_t	Code of data in packet	0xfe00
4	2	uint16_t	Access mode	0x0000
6	2	uint16_t	CRC-16 (see appendix 1)	

#### Format of answer frame (from modem to host)

Offset	Size (bytes)	Type	Description	Value
0	1	uint8_t	Address of modem	0xff
1	1	uint8_t	Type of packet	0x03
2	1	uint8_t	Number of bytes of data transmitting	0x08
3	1	uint8_t	Minor version of firmware	
4	1	uint8_t	Major version of firmware	
5	3	3 bytes	Reserved	
8	1	uint8_t	Device type ID	
9	2	uint16_t	Reserved	
11	2	uint16_t	CRC-16 (see appendix 1)	

Format of error reply is described in Appendix 2.

### 3.4.12 Reading user data

#### Supported hardware:

Super-Beacon:	not supported
Industrial Super-Beacon:	not supported
Modem HW5.1:	supported
Super-Modem:	supported
Mini-Rx (Badge, Helmet, etc.):	not supported
Mini-Tx:	not supported
Modem HW4.9:	supported
Beacon HW4.9:	not supported
Beacon HW4.5:	not supported

#### Format of request frame (from host to modem)

Offset	Size (bytes)	Type	Description	Value
0	1	uint8_t	Address of modem	0xff
1	1	uint8_t	Type of packet	0x03
2	2	uint16_t	Code of data in packet	0x0004
4	2	uint16_t	Access mode	0x0000
6	2	uint16_t	CRC-16 (see appendix 1)	

#### Format of answer frame (from modem to host)

Offset	Size (bytes)	Type	Description	Value
0	1	uint8_t	Address of modem	0xff
1	1	uint8_t	Type of packet	0x03
2	1	uint8_t	Number of bytes of data transmitting	0x84
3	1	uint8_t	Total user data size	
4	3	3 bytes	Reserved (0)	
7	128	uint8_t	User data from hedgehogs	
135	2	uint16_t	CRC-16 (see appendix 1)	

Format of error reply is described in Appendix 2.

User data from hedgehogs is the sequence of records with following structure:

Offset	Size (bytes)	Type	Description	Value
0	1	uint8_t	Address of hedgehog	H
1	1	uint8_t	Number of bytes of user data from hedgehog	M
2	M	uint8_t	M bytes of data from hedgehog H	

### 3.4.13 Writing manual device location

#### Supported hardware:

Super-Beacon:	not supported
Industrial Super-Beacon:	not supported
Modem HW5.1:	supported
Super-Modem:	supported
Mini-Rx (Badge, Helmet, etc.):	not supported
Mini-Tx:	not supported
Modem HW4.9:	supported
Beacon HW4.9:	not supported
Beacon HW4.5:	not supported

#### Format of request frame (from host to modem)

Offset	Size (bytes)	Type	Description	Value
0	1	uint8_t	Address of beacon	0x01...0xfe
1	1	uint8_t	Type of packet	0x10
2	2	uint16_t	Code of data in packet	0x5003
4	2	uint16_t	Access mode	0x0002
6	1	uint8_t	Number of bytes of data transmission	0x20
7	32	structure	Data structure (see below)	
39	2	uint16_t	CRC-16 (see appendix 1)	

#### Format of answer frame (from modem to host)

Offset	Size (bytes)	Type	Description	Value
0	1	uint8_t	Address of modem	0xff
1	1	uint8_t	Type of packet	0x10
2	2	uint16_t	Code of data	0x5003
4	2	uint16_t	reserved	
6	2	uint16_t	CRC-16 (see appendix 1)	

Format of error reply is described in Appendix 2.

#### Format of data structure:

Offset	Size (bytes)	Type	Description	Value
0	4	int32_t	X coordinate, mm	
4	4	int32_t	Y coordinate, mm	
8	4	int32_t	Z coordinate, mm	
12	1	uint8_t	Not explained	0xff
13	4	int32_t	Not explained	0
17	4	int32_t	Not explained	0
21	4	int32_t	Not explained	0
25	1	uint8_t	Not explained	0x02
26	6	6 bytes	Reserved	0



### 3.4.14 Writing manual distance between beacons

#### Supported hardware:

Super-Beacon:	not supported
Industrial Super-Beacon:	not supported
Modem HW5.1:	supported
Super-Modem:	supported
Mini-Rx (Badge, Helmet, etc.):	not supported
Mini-Tx:	not supported
Modem HW4.9:	supported
Beacon HW4.9:	not supported
Beacon HW4.5:	not supported

#### Format of request frame (from host to modem)

Offset	Size (bytes)	Type	Description	Value
0	1	uint8_t	Address of modem	0xff
1	1	uint8_t	Type of packet	0x10
2	2	uint16_t	Code of data in packet	0x4003
4	2	uint16_t	Access mode	0x0000
6	1	uint8_t	Number of bytes of data transmission	0x10
7	16	structure	Data structure (see below)	
23	2	uint16_t	CRC-16 (see appendix 1)	

#### Format of answer frame (from modem to host)

Offset	Size (bytes)	Type	Description	Value
0	1	uint8_t	Address of modem	0xff
1	1	uint8_t	Type of packet	0x10
2	2	uint16_t	Code of data	0x4003
4	2	uint16_t	reserved	
6	2	uint16_t	CRC-16 (see appendix 1)	

Format of error reply is described in Appendix 2.

#### Format of data structure:

Offset	Size (bytes)	Type	Description	Value
0	1	uint8_t	Address of first beacon	
1	1	uint8_t	Address of second beacon	
2	4	uint32_t	Distance between beacons, mm	
6	10	10 bytes	Reserved	0

## 3.5 Marvelmind API

Marvelmind API library is used by Marvelmind Dashboard software and provides interface to user's software. API is coming as dynamic-link library (DLL) for MS Windows and shared library for Linux (x86 and ARM platforms). The API implements [the communication protocol with modem](#) described in previous chapter.

In addition to the API library, the package includes C example software, which was used for testing of the API and includes calls of all API functions.

The example can be used as a basis for developing of a user's software and for porting API library interface (file 'marvelmind\_api.c') to other programming languages.

**Tested on:**

1. MS Windows XP; CPU: Intel Core 2 Duo
2. Ubuntu 16.04; CPU: Intel Core i5 3.1 GHz
3. Raspbian (2018-11-13-raspbian-stretch-full); Platform: Raspberry Pi 3 Model B+

### 3.5.1 Installation for Windows

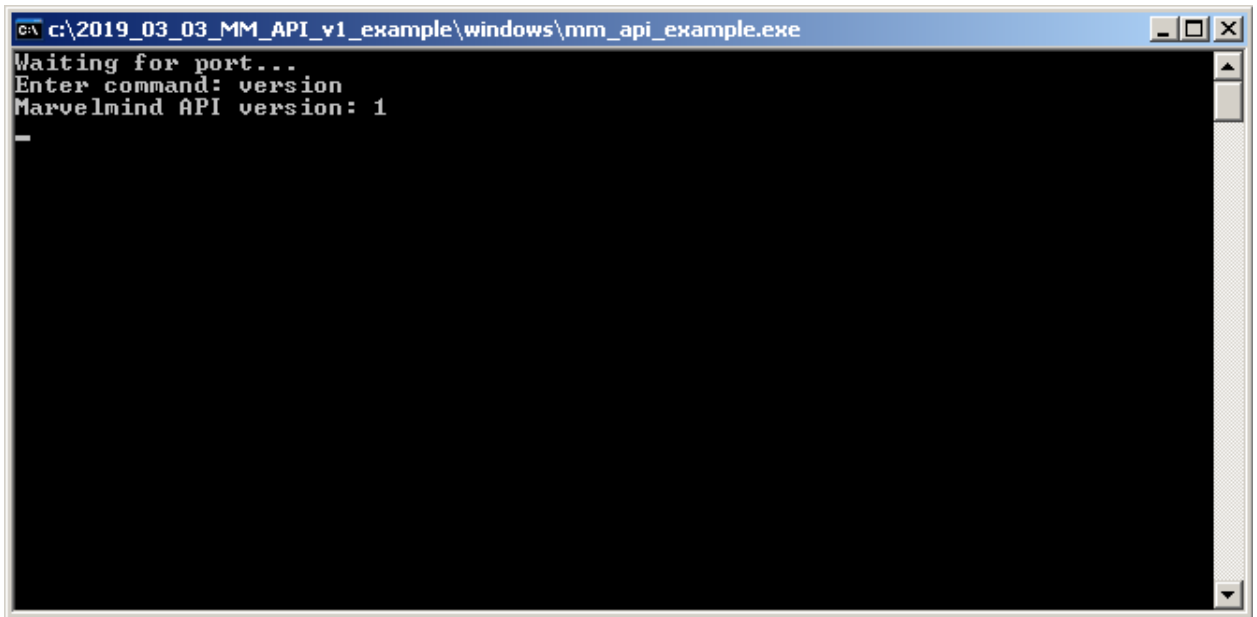
- Download Marvelmind API software package. Copy Dashboard API and example software to directory that you will use for the program. Beacons the Windows version of the example is coming with prebuilt executable file, you can immediately run 'mm\_api\_example.exe' from the 'windows' directory coming in API software package.

### 3.5.2 Installation for Linux

- Download Marvelmind API software package. Copy Dashboard API to directory that you will use for the program. Note the Linux version is provided for two hardware platforms: **x86** (most of laptops based on Intel or AMD CPU) and **arm** (for example, single-board computers like Raspberry PI)
- Copy library **libdashapi.so** corresponding to your platform to the directory **/usr/local/lib** by executing command **sudo cp libdashapi.so /usr/local/lib** in terminal opened in directory with **libdashapi.so**. After that, execute **sudo ldconfig** in terminal.
- May be, you will need to give rights for your user to access serial port by adding him to **dialout** group:
  - Execute in terminal: **sudo adduser \$USER dialout**
  - Add to the directory **/etc/udev/rules.d** file **"99-tty.rules"** with following content:  
#Marvelmind serial port rules  
KERNEL=="ttyACM0",GROUP="dialout",MODE="666"
- Build the example software – execute 'make all' in terminal opened in 'source' directory coming with the package
- Run the example by typing './mm\_api\_example' in terminal

### 3.5.3 Check connection to API

After running example software, press “space” button in terminal, type command ‘version’ and press enter. If the example software prints version of API, it can communicate with API library.



```
c:\2019_03_03_MM_API_v1_example\windows\mm_api_example.exe
Waiting for port...
Enter command: version
Marvelmind API version: 1
```

### 3.5.4 Marvelmind API library description

API is coming as dynamic-link library (DLL) for MS Windows and shared library for Linux (x86 and ARM platforms). The library includes set of functions for monitoring and controlling Marvelmind system via modem connected to USB port of the computer. This section of document contains description of all these functions.

To provide more compatibility with different programming languages, most of complex data structures are passing via untyped pointers to memory. Functions description include offset of every data field in the memory pool. In the file 'marvelmind\_api.c' from the example software you can see implementation of moving data between memory pools and fields in C structures.

Types of parameters in the description are shown in C syntax. Here is description of the types:

Type	Size (bytes)	Description
bool	1	Boolean type. Zero means false, non-zero means true
uint8_t	1	Unsigned integer value, 0...255
int8_t	1	Signed integer value in two's complement format, -128...127
uint16_t	2	Unsigned integer value, 0...65535
int16_t	2	Signed integer value in two's complement format, -32768...32767
uint32_t	4	Unsigned integer value, 0...4294967295
int32_t	4	Signed integer value in two's complement format, -2147483648...2147483647
void *	4/8	Memory pointer (address in memory). 4 bytes for 32-bit platforms, 8 bytes for 64-bit platforms.

Each function description includes set of API versions where this function is available. New API versions will support more functions for new features in Marvelmind system. Now not all features available in Dashboard are available via API, so if you need more API functions please ask to [info@marvelmind.com](mailto:info@marvelmind.com).

## List of supported functions:

Function	API versions	License needed
<a href="#">Get version of Marvelmind API library</a>	V1+	none
<a href="#">Get last error</a>	V6+	none
<a href="#">Try to open serial port</a>	V1+	none
<a href="#">Try to open serial port by given name</a>	V2+	none
<a href="#">Close serial port</a>	V1+	none
<a href="#">Get version and CPU ID of Marvelmind device</a>	V1+	none
<a href="#">Get list of devices</a>	V1+	none
<a href="#">Wake device</a>	V1+	none
<a href="#">Send device to sleep</a>	V1+	none
<a href="#">Get telemetry data from beacon</a>	V1+	none
<a href="#">Get latest location data</a>	V1+	none
<a href="#">Get latest location data (with angle)</a>	V3+	none
<a href="#">Set location of the beacon</a>	V3+	<a href="#">SW Pack v7.1xx</a>
<a href="#">Set distance between beacons</a>	V4+	<a href="#">SW Pack v7.1xx</a>
<a href="#">Get latest raw distances data</a>	V1+	none
<a href="#">Get height of the hedgehog</a>	V4+	none
<a href="#">Set height of the hedgehog</a>	V4+	<a href="#">SW Pack v7.1xx</a>
<a href="#">Get height of stationary beacon in submap</a>	V4+	none
<a href="#">Set height of stationary beacon in submap</a>	V4+	<a href="#">SW Pack v7.1xx</a>
<a href="#">Get location update rate setting</a>	V1+	none
<a href="#">Set location update rate setting</a>	V1+	<a href="#">SW Pack v7.1xx</a>
<a href="#">Add submap</a>	V1+	<a href="#">SW Pack v7.1xx</a>
<a href="#">Delete submap</a>	V1+	<a href="#">SW Pack v7.1xx</a>
<a href="#">Freeze submap</a>	V1+	<a href="#">SW Pack v7.1xx</a>
<a href="#">Unfreeze submap</a>	V1+	<a href="#">SW Pack v7.1xx</a>
<a href="#">Get submap settings</a>	V1+	none
<a href="#">Set submap settings</a>	V1+	<a href="#">SW Pack v7.1xx</a>
<a href="#">Freeze map</a>	V4+	<a href="#">SW Pack v7.1xx</a>
<a href="#">Unfreeze map</a>	V4+	<a href="#">SW Pack v7.1xx</a>
<a href="#">Get ultrasonic settings of the beacon</a>	V1+	none
<a href="#">Set ultrasonic settings of the beacon</a>	V1+	<a href="#">SW Pack v7.1xx</a>
<a href="#">Erase map</a>	V1+	<a href="#">SW Pack v7.1xx</a>
<a href="#">Reset device to default settings</a>	V1+	<a href="#">SW Pack v7.1xx</a>
<a href="#">Connect beacons to axes</a>	V2+	<a href="#">SW Pack v7.1xx</a>
<a href="#">Read modem's configuration memory dump</a>	V3+	<a href="#">SW Pack v7.1xx</a>
<a href="#">Write modem's configuration memory dump</a>	V3+	<a href="#">SW Pack v7.1xx</a>
<a href="#">Get temperature of air setting from modem</a>	V3+	none
<a href="#">Set temperature of air setting in modem</a>	V3+	none
<a href="#">Software reset of the device</a>	V3+	none
<a href="#">Get beacon real-time player settings</a>	V6+	none
<a href="#">Set beacon real-time player settings</a>	V6+	<a href="#">SW Pack v7.1xx</a>
<a href="#">Get georeferencing settings</a>	V6+	none
<a href="#">Set georeferencing settings</a>	V6+	<a href="#">SW Pack v7.1xx</a>
<a href="#">Get mode of updating positions</a>	V6+	none
<a href="#">Set mode of updating positions</a>	V6+	<a href="#">SW Pack v7.1xx</a>
<a href="#">Command to update positions</a>	V6+	<a href="#">SW Pack v7.1xx</a>
<a href="#">Check if the device type is modem</a>	V1+	none
<a href="#">Check if the device type is stationary beacon</a>	V1+	none

<a href="#">Check if the device type is hedgehog</a>	V1+	none
--	-----	------



### 3.5.4.1 Get version of Marvelmind API library

Reads version of the API library. Required to ensure the needed functions are available in this version of library.

Function name: **mm\_api\_version**

Declaration in C: `bool mm_api_version(void *pdata);`

Available for API versions: V1+

License required: none

Returned value:

Type	Description
bool	true – function successfully executed false – error in execution

Parameters:

Type	Description
void *	Pointer to data to be filled

Structure of data returned via pointer.

Type	Description
uint32_t	Version of API library

### 3.5.4.2 Get last error

Reads status of last operation with API library to differ causes of the error.

Function name: **mm\_get\_last\_error**

Declaration in C: `bool mm_get_last_error(void *pdata);`

Available for API versions: V6+

License required: none

Returned value:

Type	Description
bool	true – function successfully executed false – error in execution

Parameters:

Type	Description
void *	Pointer to data to be filled

Structure of data returned via pointer.

Type	Description
uint32_t	Status of last operation: 0: operation successfully executed 1: communication error 2: error opening serial port 3: license is required

### 3.5.4.3 Open serial port

Opens port where Marvelmind device (modem or beacon) is connected via USB (virtual serial port). You don't need to specify serial port name, because the API searching all serial ports and checks whether it corresponds to Marvelmind device or no.

Function name: **mm\_open\_port**

Declaration in C: `bool mm_open_port ();`

Available for API versions: V1+

License required: none

Returned value:

Type	Description
bool	true – function successfully executed, port is opened false – error in execution

Parameters: none

#### 3.5.4.4 Open serial port by given name

Opens port where Marvelmind device (modem or beacon) is connected via USB (virtual serial port). Function tries to open port with specified name.

Function name: **mm\_open\_port\_by\_name**

Declaration in C: `bool mm_open_port_by_name();`

Available for API versions: V2+

License required: none

Returned value:

Type	Description
bool	true – function successfully executed, port is opened false – error in execution

Parameters:

Type	Description
void *	Pointer to serial portname – sequence of ASCII characters terminated by zero (ASCII\0)

### 3.5.4.5 Close serial port

Closes port, if it was previously opened by [mm\\_open\\_port](#) function.

Function name: **mm\_close\_port**

Declaration in C: `bool mm_close_port ();`

Available for API versions: V1+

License required: none

Returned value:

Type	Description
bool	true – function successfully executed, port is closed false – error in execution

Parameters: none

### 3.5.4.6 Get version and CPU ID of Marvelmind device

Reads version and CPU ID. Version includes information about firmware version and type of device hardware. CPU ID is the unique ID of the device item.

Function name: **mm\_get\_device\_version\_and\_id**

Declaration in C: `bool mm_get_device_version_and_id (uint8_t address, void *pdata);`

Available for API versions: V1+

License required: none

Returned value:

Type	Description
bool	true – function successfully executed, version and CPU ID data retrieved false – error in execution

Parameters:

Type	Description
uint8_t	Address of Marvelmind device (1...254)
void *	Pointer to data to be filled

Structure of data returned via pointer:

Type	Description
uint8_t	Major version of firmware (example: "6", for version V6.07a)
uint8_t	Minor version of firmware (example: "7", for version V6.07a)
uint8_t	Second minor version of firmware (example: "1", for version V6.07a)
uint8_t	Device type ID (see <a href="#">appendix</a> ).
uint8_t	Firmware options (TBD).
uint32_t	CPU ID. Printing this value as hexadecimal gives CPU ID in form shown in dashboard and on the stickers on devices.

### 3.5.4.7 Get list of devices

Reads list of Marvelmind devices known to modem. The list includes list of all devices connected by radio to modem's network, including sleeping devices.

Function name: **mm\_get\_devices\_list**

Declaration in C: **bool mm\_get\_devices\_list (void \*pdata);**

Available for API versions: V1+

License required: none

Returned value:

Type	Description
bool	true – function successfully executed, list of devices is retrieved false – error in execution

Parameters:

Type	Description
void *	Pointer to data to be filled

Structure of data returned via pointer:

Type	Description
uint8_t	Number of following devices in the list (N)
N*9 bytes	Sequence of N devices structures, described in next table

Structure of each device in the list:

Type	Description
uint8_t	Address of device
bool	true = duplicated address - more than 1 device with same address was found false = not duplicated address
bool	true = device is sleeping false = device not sleeping
uint8_t	Major version of firmware (example: "6", for version V6.07a)
uint8_t	Minor version of firmware (example: "7", for version V6.07a)
uint8_t	Second minor version of firmware (example: "1", for version V6.07a)
uint8_t	Device type ID (see <a href="#">appendix</a> ).
uint8_t	Firmware options (TBD).
uint8_t	Flags: Bit 0: 1 – device connection complete – device has confirmed connection 0 – waiting for confirmation from device (like 'Connecting...' in dashboard). Bit 1...7 - TBD

### 3.5.4.8 Wake device

Sends command to wake specified device. If wake command was sent and such device is existing, the device will connect to modem in several seconds and will appear in [devices list](#).

Function name: **mm\_wake\_device**

Declaration in C: `bool mm_wake_device (uint8_t address);`

Available for API versions: V1+

License required: none

Returned value:

Type	Description
bool	true – function successfully executed, wake command was sent false – error in execution

Parameters:

Type	Description
uint8_t	1...254 - address of Marvelmind device to wake 0 – wake all devices



### 3.5.4.9 Send device to sleep

Send to sleep existing device.

Function name: **mm\_send\_to\_sleep\_device**

Declaration in C: `bool mm_send_to_sleep_device (uint8_t address);`

Available for API versions: V1+

License required: none

Returned value:

Type	Description
bool	true – function successfully executed, sleep command was sent false – error in execution

Parameters:

Type	Description
uint8_t	1...254 - address of Marvelmind device to sleep 0 – send to sleep all devices

### 3.5.4.10 Get telemetry data from beacon

Reads telemetry data of Marvelmind beacon.

Function name: **mm\_get\_beacon\_telemetry**

Declaration in C: `bool mm_get_beacon_telemetry (uint8_t address, void *pdata);`

Available for API versions: V1+

License required: none

Returned value:

Type	Description
bool	true – function successfully executed, telemetry is retrieved false – error in execution

Parameters:

Type	Description
uint8_t	Address of Marvelmind beacon (1...254)
void *	Pointer to data to be filled

Structure of data returned via pointer:

Type	Description
uint32_t	Working time of the beacon, seconds (time from reset or waking up).
int8_t	RSSI, dBm – radio signal strength
int8_t	Measured temperature, °C
uint16_t	Supply voltage, mV
16 bytes	Reserved (0)

### 3.5.4.11 Get latest location data

Reads latest updated coordinates pack from modem. Also reads user payload data if available.

Function name: **mm\_get\_last\_locations**

Declaration in C: `bool mm_get_last_locations(void *pdata);`

Available for API versions: V1+

License required: none

Returned value:

Type	Description
bool	true – function successfully executed, location data was retrieved false – error in execution

Parameters:

Type	Description
void *	Pointer to data to be filled

Structure of data returned via pointer:

Type	Description
18*6 bytes	6 18-byte data structures of last updated coordinates, see table below
bool	true – new raw distances are available to read
5 bytes	TBD
uint8_t	User payload data size (M)
M bytes	User payload data

Structure of each location data item:

Type	Description
uint8_t	Address of device (1...254) 0 - this data item is not filled
uint8_t	Head index (TBD)
int32_t	X coordinate, mm
int32_t	Y coordinate, mm
int32_t	Z coordinate, mm
uint8_t	Status flags (TBD)
uint8_t	Quality of positioning, 0...100%
uint8_t	TBD
uint8_t	TBD

### 3.5.4.12 Get latest location data (with angle)

Reads latest updated coordinates pack from modem (with angle for paired beacons). Also reads user payload data if available.

Function name: **mm\_get\_last\_locations2**

Declaration in C: `bool mm_get_last_locations2(void *pdata);`

Available for API versions: V3+

License required: none

Returned value:

Type	Description
bool	true – function successfully executed, location data was retrieved false – error in execution

Parameters:

Type	Description
void *	Pointer to data to be filled

Structure of data returned via pointer:

Type	Description
20*6 bytes	6 20-byte data structures of last updated coordinates, see table below
bool	true – new raw distances are available to read
5 bytes	TBD
uint8_t	User payload data size (M)
M bytes	User payload data

Structure of each location data item:

Type	Description
uint8_t	Address of device (1...254) 0 - this data item is not filled
uint8_t	Head index (TBD)
int32_t	X coordinate, mm
int32_t	Y coordinate, mm
int32_t	Z coordinate, mm
uint8_t	Status flags (TBD)
uint8_t	Quality of positioning, 0...100%
uint8_t	TBD
uint8_t	TBD
uint16_t	Bit 0...11 – angle of rotation in 1/10 degree (if paired beacons feature is enabled) Bit 12 – 1 = angle not available Bit 13...15 - reserved

### 3.5.4.13 Set location of the beacon

Manual setup of location of the specified beacon.

Function name: **mm\_set\_beacon\_location**

Declaration in C: `bool mm_set_beacon_location (uint8_t address, void *pdata);`

Available for API versions: V3+

License required: [SW Pack v7.1xx](#)

Returned value:

Type	Description
bool	true – function successfully executed, location is updated false – error in execution

Parameters:

Type	Description
uint8_t	Address of the beacon
void *	Pointer to buffer with location data

Structure of data by pointer (should be filled before function call):

Type	Description
int32_t	New X coordinate of the beacon, mm
int32_t	New Y coordinate of the beacon, mm
int32_t	New Z coordinate of the beacon, mm

### 3.5.4.14 Set distance between beacons

Manual setup of distance between beacons.

Function name: **mm\_set\_beacons\_distance**

Declaration in C: `bool mm_set_beacons_distance (void *pdata);`

Available for API versions: V4+

License required: [SW Pack v7.1xx](#)

Returned value:

Type	Description
bool	true – function successfully executed, distance is written false – error in execution

Parameters:

Type	Description
void *	Pointer to buffer with distance data

Structure of data by pointer (should be filled before function call):

Type	Description
uint8_t	Address of first beacon
uint8_t	Address of second beacon
int32_t	Distance between beacons, mm

### 3.5.4.15 Get latest raw distances data

Reads latest updated raw distances pack from modem.

Function name: **mm\_get\_last\_distances**

Declaration in C: `bool mm_get_last_distances(void *pdata);`

Available for API versions: V1+

License required: none

Returned value:

Type	Description
bool	true – function successfully executed, raw distances data was retrieved false – error in execution

Parameters:

Type	Description
void *	Pointer to data to be filled

Structure of data returned via pointer:

Type	Description
uint8_t	Number of raw distances data items (N). Maximum number of raw distances per request is 16: $N \leq 16$
9*N bytes	N 9-byte data structures of last raw distances, see table below

Structure of each raw distance data item:

Type	Description
uint8_t	Address of ultrasonic RX device (1...254) 0 - this data item is not filled
uint8_t	RX Head index (TBD)
uint8_t	Address of ultrasonic TX device (1...254) 0 - this data item is not filled
uint8_t	TX Head index (TBD)
uint32_t	Distance from TX device to RX device, mm
uint8_t	TBD

### 3.5.4.16 Get height of the hedgehog

Returns height of mobile beacon (hedgehog).

Function name: **mm\_get\_hedge\_height**

Declaration in C: `bool mm_get_hedge_height (uint8_t address, void *pdata);`

Available for API versions: V4+

License required: none

Returned value:

Type	Description
bool	true – function successfully executed, height is returned false – error in execution

Parameters:

Type	Description
uint8_t	Address of the hedgehog
void *	Pointer to buffer with height data

Structure of data by pointer:

Type	Description
int32_t	Height of the hedgehog, mm



### 3.5.4.17 Set height of the hedgehog

Setup height of mobile beacon (hedgehog).

Function name: **mm\_set\_hedge\_height**

Declaration in C: `bool mm_set_hedge_height (uint8_t address, void *pdata);`

Available for API versions: V4+

License required: [SW Pack v7.1xx](#)

Returned value:

Type	Description
bool	true – function successfully executed, height is changed false – error in execution

Parameters:

Type	Description
uint8_t	Address of the hedgehog
void *	Pointer to buffer with height data

Structure of data by pointer (should be filled before function call):

Type	Description
int32_t	Height of the hedgehog, mm

### 3.5.4.18 Get height of the stationary beacon in submap

Returns height of stationary beacon in submap.

Function name: **mm\_get\_beacon\_height**

Declaration in C: `bool mm_get_beacon_height (uint8_t address, void *pdata);`

Available for API versions: V4+

License required: none

Returned value:

Type	Description
bool	true – function successfully executed, height is returned false – error in execution

Parameters:

Type	Description
uint8_t	Address of the beacon
void *	Pointer to buffer with height data

Structure of data by pointer:

Type	Description
uint8_t	Submap ID, should be filled before function call
int32_t	Height of the beacon, mm

### 3.5.4.19 Set height of the stationary beacon in submap

Setup height of stationary beacon in submap.

Function name: **mm\_set\_beacon\_height**

Declaration in C: `bool mm_set_beacon_height (uint8_t address, void *pdata);`

Available for API versions: V4+

License required: [SW Pack v7.1xx](#)

Returned value:

Type	Description
bool	true – function successfully executed, height is changed false – error in execution

Parameters:

Type	Description
uint8_t	Address of the beacon
void *	Pointer to buffer with height data

Structure of data by pointer (should be filled before function call):

Type	Description
uint8_t	Submap ID
int32_t	Height of the beacon, mm

### 3.5.4.20 Get location update rate setting

Reads location update rate setting from modem.

Function name: **mm\_get\_update\_rate\_setting**

Declaration in C: `bool mm_get_update_rate_setting (void *pdata);`

Available for API versions: V1+

License required: none

Returned value:

Type	Description
bool	true – function successfully executed, update rate was retrieved false – error in execution

Parameters:

Type	Description
void *	Pointer to data to be filled

Structure of data returned via pointer:

Type	Description
uint32_t	Location update rate setting in mHz. So, 1000 is returned for 1 Hz, 16000 for 16 Hz, 50 for 0.05 Hz mode.

### 3.5.4.21 Set location update rate setting

Writes location update rate setting to modem.

Function name: **mm\_set\_update\_rate\_setting**

Declaration in C: `bool mm_set_update_rate_setting (void *pdata);`

Available for API versions: V1+

License required: [SW Pack v7.1xx](#)

Returned value:

Type	Description
bool	true – function successfully executed, update rate was changed false – error in execution

Parameters:

Type	Description
void *	Pointer to data

Structure of data by pointer (should be filled before function call):

Type	Description
uint32_t	Location update rate setting in mHz. So, 1000 is returned for 1 Hz, 16000 for 16 Hz, 50 for 0.05 Hz mode. The system will use most close to specified update rate from the series: 0.05 Hz, 0.1 Hz, 0.2 Hz, 0.5Hz, 1 Hz, 2 Hz, 4 Hz, 8 Hz, 12 Hz, 16 Hz, 16+Hz.

### 3.5.4.22 Add submap

Adds new submap.

Function name: **mm\_add\_submap**

Declaration in C: `bool mm_add_submap (uint8_t submapId);`

Available for API versions: V1+

License required: [SW Pack v7.1xx](#)

Returned value:

Type	Description
bool	true – function successfully executed, submap was added false – error in execution

Parameters:

Type	Description
uint8_t	Submap ID to add (0...254)

### 3.5.4.23 Delete submap

Delete existing submap.

Function name: **mm\_delete\_submap**

Declaration in C: `bool mm_delete_submap (uint8_t submapId);`

Available for API versions: V1+

License required: [SW Pack v7.1xx](#)

Returned value:

Type	Description
bool	true – function successfully executed, submap was removed false – error in execution

Parameters:

Type	Description
uint8_t	Submap ID to delete (0...254)

### 3.5.4.24 Freeze submap

Freezes submap.

Function name: **mm\_freeze\_submap**

Declaration in C: `bool mm_freeze_submap (uint8_t submapId);`

Available for API versions: V1+

License required: [SW Pack v7.1xx](#)

Returned value:

Type	Description
bool	true – function successfully executed, submap is frozen false – error in execution

Parameters:

Type	Description
uint8_t	Submap ID to freeze (0...254)



### 3.5.4.25 Unfreeze submap

Unfreezes submap.

Function name: **mm\_unfreeze\_submap**

Declaration in C: `bool mm_unfreeze_submap (uint8_t submapId);`

Available for API versions: V1+

License required: [SW Pack v7.1xx](#)

Returned value:

Type	Description
bool	true – function successfully executed, submap is unfrozen false – error in execution

Parameters:

Type	Description
uint8_t	Submap ID to unfreeze (0...254)

### 3.5.4.26 Get submap settings

Reads submap settings from modem.

Function name: **mm\_get\_submap\_settings**

Declaration in C: `bool mm_get_submap_settings (uint8_t submapId , void *pdata);`

Available for API versions: V1+

License required: none

Returned value:

Type	Description
bool	true – function successfully executed, submap settings were retrieved false – error in execution

Parameters:

Type	Description
uint8_t	Submap ID (0...254)
void *	Pointer to data to be filled

Structure of data returned via pointer:

Type	Description
uint8_t	Starting beacon trilateration
uint8_t	Starting set of beacons, beacon 1
uint8_t	Starting set of beacons, beacon 2
uint8_t	Starting set of beacons, beacon 3
uint8_t	Starting set of beacons, beacon 4
bool	true = 3D navigation enabled
bool	true = Submap is used only for Z coordinate
bool	true = manual limitation distance false = auto limitation distance
uint8_t	Maximum distance, meters (for manual limitation distances)
int16_t	Submap X shift, cm
int16_t	Submap Y shift, cm
int16_t	Submap Z shift, cm
uint16_t	Submap rotation, centidegrees
int16_t	Plane rotation quaternion, W (quaternion is normalized to 10000)
int16_t	Plane rotation quaternion, X
int16_t	Plane rotation quaternion, Y
int16_t	Plane rotation quaternion, Z
int16_t	Service zone thickness, cm
int16_t	Hedges height in 2D mode
bool	true = submap is frozen
bool	true = submap is locked
bool	true = stationary beacons are higher than mobile
bool	true = submap is mirrored
4 bytes	List of addresses of beacons in submap (0 = none)
8 bytes	List of ID's of nearby submaps (255 = none)
uint8_t	Number of service zone polygon points (P)
P*4 bytes	List of service zone polygon points structures (see below)

Structure of service zone polygon point:

Type	Description
int16_t	X, cm
int16_t	Y, cm

### 3.5.4.27 Set submap settings

Writes submap settings to modem.

Function name: **mm\_set\_submap\_settings**

Declaration in C: `bool mm_set_submap_settings (uint8_t submapId , void *pdata);`

Available for API versions: V1+

License required: [SW Pack v7.1xx](#)

Returned value:

Type	Description
bool	true – function successfully executed, submap settings were changed false – error in execution

Parameters:

Type	Description
uint8_t	Submap ID (0...254)
void *	Pointer to data to be written (see ' <a href="#">get_submap_settings</a> ' function).

### 3.5.4.28 Freeze map

Freezes submap.

Function name: **mm\_freeze\_map**

Declaration in C: `bool mm_freeze_map ();`

Available for API versions: V4+

License required: [SW Pack v7.1xx](#)

Returned value:

Type	Description
bool	true – function successfully executed, map is frozen false – error in execution

### 3.5.4.29 Unfreeze map

Freezes submap.

Function name: **mm\_unfreeze\_map**

Declaration in C: `bool mm_freeze_map ();`

Available for API versions: V4+

License required: [SW Pack v7.1xx](#)

Returned value:

Type	Description
bool	true – function successfully executed, map is unfrozen false – error in execution

### 3.5.4.30 Get ultrasonic settings of the beacon

Reads ultrasonic settings from specified beacon.

Function name: **mm\_get\_ultrasound\_settings**

Declaration in C: `bool mm_get_ultrasound_settings (uint8_t address , void *pdata);`

Available for API versions: V1+

License required: none

Returned value:

Type	Description
bool	true – function successfully executed, ultrasonic settings were retrieved false – error in execution

Parameters:

Type	Description
uint8_t	Address of the beacon (1...254)
void *	Pointer to data to be filled

Structure of data returned via pointer:

Type	Description
uint16_t	Frequency of ultrasound TX (not relevant for DSP RX-only beacons)
uint8_t	Number of TX periods (not relevant for DSP RX-only beacons)
bool	true= use AGC for RX false = manual gain for RX
uint16_t	Manual gain value (0...4000)
bool	true = Sensor RX1 is enabled in normal mode
bool	true = Sensor RX2 is enabled in normal mode
bool	true = Sensor RX3 is enabled in normal mode
bool	true = Sensor RX4 is enabled in normal mode
bool	true = Sensor RX5 is enabled in normal mode
bool	true = Sensor RX1 is enabled in frozen mode
bool	true = Sensor RX2 is enabled in frozen mode
bool	true = Sensor RX3 is enabled in frozen mode
bool	true = Sensor RX4 is enabled in frozen mode
bool	true = Sensor RX5 is enabled in frozen mode
uint8_t	Index of DSP RX filter (relevant only for DSP beacons) 0 = 19 kHz 1 = 25 kHz 2 = 31 kHz 3 = 37 kHz 4 = 45 kHz

### 3.5.4.31 Set ultrasonic settings of the beacon

Write ultrasonic settings to specified beacon.

Function name: **mm\_set\_ultrasound\_settings**

Declaration in C: `bool mm_set_ultrasound_settings (uint8_t address, void *pdata);`

Available for API versions: V1+

License required: [SW Pack v7.1xx](#)

Returned value:

Type	Description
bool	true – function successfully executed, ultrasonic settings were changed false – error in execution

Parameters:

Type	Description
uint8_t	Address of the beacon (1...254)
void *	Pointer to data to be written (see ' <a href="#">get ultrasonic settings</a> ' function).

### 3.5.4.32 Erase map

Erase map in modem – remove all submaps (except submap 0), reset submap 0 to initial state, remove all connected beacons from network.

Function name: **mm\_erase\_map**

Declaration in C: `bool mm_erase_map ();`

Available for API versions: V1+

License required: [SW Pack v7.1xx](#)

Returned value:

Type	Description
bool	true – function successfully executed, map erased false – error in execution

Parameters: none



### 3.5.4.33 Reset device to default settings

Reset device to default settings (radio, ultrasonic etc).

Function name: **mm\_set\_default\_settings**

Declaration in C: `bool mm_set_default_settings (uint8_t address);`

Available for API versions: V1+

License required: [SW Pack v7.1xx](#)

Returned value:

Type	Description
bool	true – function successfully executed, device was reset to default settings
	false – error in execution

Parameters:

Type	Description
uint8_t	Address of the device (1...254)
	255 – reset to default the device connected via USB

### 3.5.4.34 Connect beacons to axes

Shift map so selected beacons will be on axes.

Function name: **mm\_beacons\_to\_axes**

Declaration in C: `bool mm_beacons_to_axes (uint8_t address_0, uint8_t address_x, uint8_t address_y);`

Available for API versions: V2+

License required: [SW Pack v7.1xx](#)

Returned value:

Type	Description
bool	true – function successfully executed, map shifted false – error in execution

Parameters:

Type	Description
uint8_t	address_0 – address of beacon which should be in the center (X=0, Y=0)
uint8_t	address_x – address of beacon which should be along X axis (Y= 0)
uint8_t	address_y – address of beacon which should be in positive direction of Y (Y>0)

### 3.5.4.35 Read dump of modem's configuration memory

Reads dump of modem's configuration memory. Allows saving modem's settings and stored map.

Function name: **mm\_read\_flash\_dump**

Declaration in C: `bool mm_read_flash_dump(uint32_t offset, uint32_t size, void *pdata);`

Available for API versions: V3+

License required: [SW Pack v7.1xx](#)

Returned value:

Type	Description
bool	true – function successfully executed, dump was read false – error in execution

Parameters:

Type	Description
uint32_t	offset – offset from start of configuration memory, bytes
uint32_t	size – size of data to read, bytes
void *	pdata – pointer to user's buffer for receiving data

### 3.5.4.36 Write dump of modem's configuration memory

Write data dump to modem's configuration memory. Allows to restore modem's settings and map.

Function name: **mm\_write\_flash\_dump**

Declaration in C: `bool mm_write_flash_dump(uint32_t offset, uint32_t size, void *pdata);`

Available for API versions: V3+

License required: [SW Pack v7.1xx](#)

Returned value:

Type	Description
bool	true – function successfully executed, dump was written false – error in execution

Parameters:

Type	Description
uint32_t	offset – offset from start of configuration memory, bytes For correct operation offset should be aligned to 4096 bytes page (value 0, 4096, 8192 and so on).
uint32_t	size – size of data to write, bytes
void *	pdata – pointer to user's buffer with data

Note: After writing the configuration, [software reset](#) of the modem (**mm\_reset\_device(255)**) is recommended to apply new settings and prevent overwriting them.

### 3.5.4.37 Restart (soft reset) of the device

Executes software reset for specified device.

Function name: **mm\_reset\_device**

Declaration in C: `bool mm_reset_device (uint8_t address);`

Available for API versions: V3+

License required: none

Returned value:

Type	Description
bool	true – function successfully executed, device is resetting false – error in execution

Parameters:

Type	Description
uint8_t	Address of the device (1...254) 255 –software reset for device connected via USB

### 3.5.4.38 Read temperature of air setting from modem

Reads temperature of air setting (in Celsius degrees) from modem.

Function name: **mm\_get\_air\_temperature**

Declaration in C: `bool mm_get_air_temperature (void *pdata);`

Available for API versions: V3+

License required: none

Returned value:

Type	Description
bool	true – function successfully executed, temperature is returned false – error in execution

Structure of data returned via pdata pointer:

Type	Description
int8_t	Temperature of air, Celsius degrees

### 3.5.4.39 Write temperature of air setting to modem

Setup temperature of air setting (in Celsius degrees) in modem.

Function name: **mm\_set\_air\_temperature**

Declaration in C: `bool mm_set_air_temperature (void *pdata);`

Available for API versions: V3+

License required: none

Returned value:

Type	Description
bool	true – function successfully executed, temperature was written false – error in execution

Structure of data which user should supply via pdata pointer:

Type	Description
int8_t	Temperature of air, Celsius degrees

#### 3.5.4.40 Get beacon real-time player settings

Reads real-time player settings for the beacon.

Function name: **mm\_get\_realtime\_player\_settings**

Declaration in C: `bool mm_get_realtime_player_settings (uint8_t address, void *pdata);`

Available for API versions: V6+

License required: none

Returned value:

Type	Description
bool	true – function successfully executed false – error in execution

Parameters:

Type	Description
uint8_t	address - address of the beacon (1...254)
void *	pdata - pointer to data to be filled

Structure of data returned via pointer:

Type	Description
bool	true = real-time player is enabled
uint8_t	Number of real-time player forward samples to process
uint8_t	Number of real-time player backward samples to process
uint8_t	Reserved (0)
uint8_t	Reserved (0)



### 3.5.4.41 Set beacon real-time player settings

Setup real-time player settings for the beacon.

Function name: **mm\_set\_realtime\_player\_settings**

Declaration in C: `bool mm_set_realtime_player_settings (uint8_t address, void *pdata);`

Available for API versions: V6+

License required: [SW Pack v7.1xx](#)

Returned value:

Type	Description
bool	true – function successfully executed false – error in execution

Parameters:

Type	Description
uint8_t	address - address of the beacon (1...254)
void *	pdata - pointer to data to write (see ' <a href="#">Get beacon real-time player settings</a> ' function)

### 3.5.4.42 Get georeferencing settings

Reads georeferencing settings (geo location of point (X=0 ,Y=0) of Marvelmind map).

Function name: **mm\_get\_georeferencing\_settings**

Declaration in C: `bool mm_get_georeferencing_settings (void *pdata);`

Available for API versions: V6+

License required: none

Returned value:

Type	Description
bool	true – function successfully executed false – error in execution

Parameters:

Type	Description
void *	pdata - pointer to data to be filled

Structure of data returned via pointer:

Type	Description
int32_t	Latitude, $\times 10^{-7}$ degrees
int32_t	Longitude, $\times 10^{-7}$ degrees

3.5.4.43 Set georeferencing settings

Setup georeferencing settings (geo location of point (X=0 ,Y=0) of Marvelmind map).

Function name: **mm\_set\_georeferencing\_settings**

Declaration in C: `bool mm_set_georeferencing_settings (void *pdata);`

Available for API versions: V6+

License required: [SW Pack v7.1xx](#)

Returned value:

Type	Description
bool	true – function successfully executed false – error in execution

Parameters:

Type	Description
void *	pdata - pointer to data to write (see <a href="#">‘Get georeferencing settings’</a> function)

#### 3.5.4.44 Get mode of updating positions

Reads current mode of updating positions of mobile beacons.

Function name: **mm\_get\_update\_position\_mode**

Declaration in C: `bool mm_get_update_position_mode (void *pdata);`

Available for API versions: V6+

License required: none

Returned value:

Type	Description
bool	true – function successfully executed false – error in execution

Parameters:

Type	Description
void *	pdata - pointer to data to be filled

Structure of data returned via pointer:

Type	Description
uint8_t	Mode of updating positions of mobile beacons: 0 – auto update positions (default mode) 1 – update positions by user request at next update cycle 2 – update positions by user request immediately
7 bytes	Reserved (0)

### 3.5.4.45 Set mode of updating positions

Setup mode of updating positions of mobile beacons.

Function name: **mm\_set\_update\_position\_mode**

Declaration in C: `bool mm_set_update_position_mode (void *pdata);`

Available for API versions: V6+

License required: [SW Pack v7.1xx](#)

Returned value:

Type	Description
bool	true – function successfully executed false – error in execution

Parameters:

Type	Description
void *	pdata - pointer to data to write (see function ' <a href="#">Get mode of updating positions</a> ')

### 3.5.4.46 Command to update positions

Send command to update positions of mobile beacons (if [update mode](#) is not automatic).

Function name: **mm\_set\_update\_position\_command**

Declaration in C: `bool mm_set_update_position_command (void *pdata);`

Available for API versions: V6+

License required: [SW Pack v7.1xx](#)

Returned value:

Type	Description
bool	true – function successfully executed false – error in execution

Parameters:

Type	Description
void *	pdata - pointer to data to write

Structure of data by pointer:

Type	Description
8 bytes	Reserved (0)

### 3.5.4.47 Check whether device type is modem

Checks whether the specified device type corresponds to modem.

Function name: **mm\_device\_is\_modem**

Declaration in C: `bool mm_device_is_modem (uint8_t deviceType);`

Available for API versions: V1+

License required: none

Returned value:

Type	Description
bool	true – specified device type corresponds to modem

Parameters:

Type	Description
uint8_t	<a href="#">Device type</a> to check

### 3.5.4.48 Check whether device type is stationary beacon

Checks whether the specified device type corresponds to stationary beacon.

Function name: **mm\_device\_is\_beacon**

Declaration in C: `bool mm_device_is_beacon (uint8_t deviceType);`

Available for API versions: V1+

License required: none

Returned value:

Type	Description
bool	true – specified device type corresponds to stationary beacon

Parameters:

Type	Description
uint8_t	<a href="#">Device type</a> to check



### 3.5.4.49 Check whether device type is hedgehog

Checks whether the specified device type corresponds to hedgehog.

Function name: **mm\_device\_is\_hedgehog**

Declaration in C: `bool mm_device_is_hedgehog (uint8_t deviceType);`

Available for API versions: V1+

License required: none

Returned value:

Type	Description
bool	true – specified device type corresponds to hedgehog

Parameters:

Type	Description
uint8_t	<a href="#">Device type</a> to check

### 3.5.4. Description of C example for Marvelmind API

C example is used for testing of Marvelmind API and can be used as basis for building of user application.

The C example is the console application. It was tested on following platforms:

- CPU: Intel Core 2 Duo, OS: MS Windows XP;
- CPU: Intel Core i5, OS: Linux Ubuntu 16.04;
- Raspberry Pi 3 Model B+, OS: Raspbian (2018-11-13-raspbian-stretch-full)

On the Windows platform the example was built with CodeBlocks IDE and so the example includes CodeBlocks project file.

On the Linux platforms, the example was built with using make utility and so the example includes makefile for this.

The example includes following modules:

File name	Description
main.c	Module with main () function. Calls of functions of example and implements simple command line interface.
marvelmind_example.c marvelmind_example.h	marvelmindStart() – initialization of the example marvelmindFinish() – called after finishing work with API marvelmindCycle() – frequently called from main loop  Also, module includes several function for processing commands entered by user.
marvelmind_api.c marvelmind_api.h	marvelmindAPILoad() – loads API library marvelmindAPIFree() – frees memory used by API library All functions of communication with API library.
marvelmind_devices.c marvelmind_devices.h	Supports list of beacons retrieved from modem by calling ' <a href="#">get devices list</a> ' command. Each beacon includes data about its location and distances to other beacons.
marvelmind_pos.c marvelmind_pos.h	Reads <a href="#">latest location data</a> and <a href="#">latest raw distances</a> . Updates these data in the devices list.
marvelmind_utils.c marvelmind_utils.h	Some helper functions used by other modules.

#### How the example works:

1. Try to [open](#) serial port until success
2. When port is opened, the program reads version of device connected via USB. If this is modem, the program continues to execute next steps
3. When connected to modem, the program reads the [devices list](#) with 1 Hz rate. The devices list is compared with currently stored in marvelmind\_devices.c module and the list in marvelmind\_devices.c is updated, if any changes are detected. All changes are printed in console
4. When connected to modem, the program reads the [latest location data](#) with 20 Hz rate. If the flag of new raw distances data is set, the program reads [latest raw distances](#). The program compares locations and distances with data in devices list in marvelmind\_devices.c and updates the data if they are changed. All changed data are printed in console
5. If the program can't get latest location data for 10 times, it [closes the port](#) and returns to step 1 – tries to open the port again. Reopening of the port is needed for cases when modem was disconnected and connected back to USB

6. If user press '**space**' button, the program shows 'Enter command: ' message and waits for user command. Most of API functions are called by user command, see below for details

#### User commands:

If user press '**space**' button when program is running, the program shows message '**Enter command:** '. User should type command on keyboard and press **enter**.

The table below contains format of all user commands:

Commands group	Description
API version	Format of command: <b>version</b> Action: Prints version of API library
Exit from program	Format of command: <b>quit</b> Action: Finishes program execution
Sleep/wake	Format of command: <b>wake &lt;address&gt;</b> Action: Execute <a href="#">wake</a> command. Examples: <b>wake 5</b> - send command to wake device 5 <b>wake 0</b> - send command to wake all devices
	Format of command: <b>sleep &lt;address&gt;</b> Action: Execute <a href="#">sending to sleep</a> command. Examples: <b>sleep 5</b> - send to sleep device 5 <b>sleep 0</b> - send to sleep all devices
Default	Format of command: <b>default &lt;address&gt;</b> Action: Execute <a href="#">reset to default settings</a> command. Examples: <b>default 5</b> - set default settings for device 5
Read telemetry	Format of command: <b>tele &lt;address&gt;</b> Action: Reads and prints <a href="#">telemetry</a> data of beacon. Examples: <b>tele 5</b> - read and print telemetry of beacon 5
Submap commands	Format of command: <b>submap add &lt;submapId&gt;</b> Action: Execute command to <a href="#">add submap</a> with specified submap ID. Example: <b>submap add 1</b> - add submap 1

	<p>Format of command:  <b>submap delete &lt;submapId&gt;</b>  Action:  Execute command to <a href="#">delete submap</a> with specified submap ID.  Example:  <b>submap delete 1</b>                      - delete submap 1</p>
	<p>Format of command:  <b>submap freeze &lt;submapId&gt;</b>  Action:  Execute command to <a href="#">freeze submap</a> with specified submap ID.  Example:  <b>submap freeze 0</b>                      - freeze submap 0</p>
	<p>Format of command:  <b>submap unfreeze &lt;submapId&gt;</b>  Action:  Execute command to <a href="#">unfreeze submap</a> with specified submap ID.  Example:  <b>submap unfreeze 0</b>                      - unfreeze submap 0</p>
	<p>Format of command:  <b>submap get &lt;submapId&gt;</b>  Action:  Execute command to <a href="#">get settings of submap</a> with specified submap ID.  Example:  <b>submap get 0</b>                      - get and print settings of submap 0</p>
	<p>Format of command:  <b>submap testset &lt;submapId&gt;</b>  Action:  Execute command to <a href="#">set settings of submap</a> with specified submap ID. The program writes some predefined settings for testing of the command; please see the example code.  Example:  <b>submap testset 0</b>                      - modify settings of submap 0</p>
Map commands	<p>Format of command:  <b>map erase</b>  Action:  Execute <a href="#">erase map</a> command.  Example:  <b>map erase</b>                      - erase map in modem</p>
	<p>Format of command:  <b>map freeze</b>  Action:  Execute <a href="#">freeze map</a> command.  Example:</p>

	<b>map freeze</b> - freeze map
	Format of command: <b>map unfreeze</b> Action: Execute <a href="#">unfreeze map</a> command. Example: <b>map unfreeze</b> - unfreeze map
Update rate commands	Format of command: <b>rate get</b> Action: Execute <a href="#">reading update rate</a> setting command. Example: <b>rate get</b> - read and print update rate setting
	Format of command: <b>rate set &lt;value&gt;</b> Action: Execute <a href="#">change update rate</a> setting command. Value is given in Hz Example: <b>rate set 0.5</b> - set update rate 0.5 Hz
Ultrasound commands	Format of command: <b>usound get &lt;address&gt;</b> Action: Execute <a href="#">reading ultrasonic settings</a> for specified beacon. Example: <b>usound get 5</b> - read and print ultrasound settings of beacon 5
	Format of command: <b>usound testset &lt;address&gt;</b> Action: Execute <a href="#">writing ultrasonic settings</a> for specified beacon. The program writes some predefined settings for testing of the command; please see the example code. Example: <b>usound testset 5</b> - modify ultrasound settings of beacon 5
Connect to axes command	Format of command: <b>axes &lt;address_0&gt; &lt;address_x&gt; &lt;address_y&gt;</b> Action: Execute <a href="#">connect beacons to axes</a> command.. Example: <b>axes 3 4 5</b> - set beacon 3 to X=0, Y=0; beacon 4 along X (Y=0) and beacon 5 above X (Y>0)

Read configuration memory dump from modem	<p>Format of command:  <b>read_dump &lt;offset&gt; &lt;size&gt;</b></p> <p>Action:  Execute <a href="#">read dump of modem configuration memory</a> command.</p> <p>Example:  <b>read_dump 0 1000</b> - read first 1000 bytes from beginning of configuration memory</p>
Write configuration memory test dump to modem	<p>Format of command:  <b>write_dump_test &lt;offset&gt; &lt;size&gt;</b></p> <p>Action:  Execute <a href="#">write dump of modem configuration memory</a> command.</p> <p>Example:  <b>write_dump_test 0 1000</b> - fills first 1000 bytes from beginning of configuration memory by test pattern</p>
Software reset of device	<p>Format of command:  <b>reset &lt;address&gt;</b></p> <p>Action:  Execute <a href="#">software reset</a> command.</p> <p>Example:  <b>reset 255</b> - executes software reset for device connected via USB</p>
Temperature of air commands	<p>Format of command:  <b>temperature get</b></p> <p>Action:  Execute reading <a href="#">temperature of air setting</a> from modem</p> <p>Example:  <b>temperature get</b> read and print ultrasound temperature of air setting</p>
	<p>Format of command:  <b>temperature set &lt;value&gt;</b></p> <p>Action:  Execute writing <a href="#">temperature of air setting</a> to modem</p> <p>Example:  <b>temperature set 30</b> setup temperature of air setting to 30 Celsius degrees</p>
Set location of beacon	<p>Format of command:  <b>setloc &lt;address&gt; &lt;X&gt; &lt;Y&gt; &lt;Z&gt;</b></p> <p>Action:  Execute <a href="#">set location of the beacon</a> command. X, Y, Z are coordinates in meters.</p> <p>Example:  <b>setloc 12 1.51 3.45 2.0</b> - sets location of beacon 12 to X= 1.51 m, Y= 3.45 m, Z= 2.0 m</p>

Set distance between beacons	<p>Format of command:  <b>setdist &lt;address1&gt; &lt;address2&gt; &lt;distance&gt;</b></p> <p>Action:  Execute <a href="#">set distance between beacons</a> command. Address1 and address2 are addresses of beacons. Distance is distance in meters.</p> <p>Example:  <b>setdist 12 13 16.5</b>     - sets distance between beacons 12 and 13 to 16.5 meters</p>
Heights commands	<p>Format of command:  <b>height_h get &lt;address&gt;</b></p> <p>Action:  Execute <a href="#">get hedge height</a> command. Address is the address of the hedgehog.</p> <p>Example:  <b>height_h get 15</b>     - reads and prints height of hedgehog 15</p>
	<p>Format of command:  <b>height_h set &lt;address&gt; &lt;height&gt;</b></p> <p>Action:  Execute <a href="#">set hedge height</a> command. Address is the address of the hedgehog. Height in meters</p> <p>Example:  <b>height_h set 15 2.5</b>     - setup height of hedgehog 15 to 2.5 meters</p>
	<p>Format of command:  <b>height_b get &lt;address&gt; &lt;submap_id&gt;</b></p> <p>Action:  Execute <a href="#">get stationary beacon height</a> command. Address is the address of the beacon. Submap_id is ID of submap where beacon belongs.</p> <p>Example:  <b>height_b get 12 0</b>     - reads and prints height of stationary beacon 12 in submap 0.</p>
	<p>Format of command:  <b>height_b set &lt;address&gt; &lt;submap_id&gt; &lt;height&gt;</b></p> <p>Action:  Execute <a href="#">set stationary beacon height</a> command. Address is the address of the hedgehog. Submap_id is ID of submap where beacon belongs. Height in meters</p> <p>Example:  <b>height_b set 12 0 5.1</b>     - setup height of beacon 12 in submap 0 to 5.1 meters</p>
Real-time player commands	<p>Format of command:  <b>rtp get &lt;address&gt;</b></p> <p>Action:  Execute <a href="#">get real-time player settings</a> command. Address is the address of the beacon.</p> <p>Example:</p>

	<p><b>rtp get 15</b> - reads and prints real-time player settings of beacon 15</p> <p>Format of command:  <b>rtp testset &lt;address&gt;</b>  Action:  Execute <a href="#">set real-time player settings</a> command. Address is the address of the beacon. The program writes some predefined settings for testing of the command; please see the example code.</p> <p>Example:  <b>rtp testset 15</b> - setup test real-time player settings for beacon 15</p>
	<p>Format of command:  <b>georef get</b>  Action:  Execute <a href="#">get georeferencing</a> settings command.  Example:  <b>georef get</b> - reads and prints georeferencing settings</p> <p>Format of command:  <b>georef set &lt;latitude&gt; &lt;longitude&gt;</b>  Action:  Execute <a href="#">set georeferencing</a> settings command.  Example:  <b>georef set 10 20</b> – write georeferencing 10 degrees latitude, 20 degrees longitude</p>
Update mode commands	<p>Format of command:  <b>update_mode get</b>  Action:  Execute <a href="#">get positions update mode</a> command.  Example:  <b>update_mode get</b> - reads and prints positions update mode</p>
	<p>Format of command:  <b>update_mode set &lt;mode&gt;</b>  Action:  Execute <a href="#">set positions update mode</a> command.  Example:  <b>update_mode set 0</b> - set automatic mode of positions update</p>
	<p>Format of command:  <b>update</b>  Action:  Execute <a href="#">update positions</a> command.  Example:  <b>update</b> - update positions of mobile beacons according to <a href="#">current mode</a></p>



### 3.5.4.50 Device types

Here is the list of 'Device type ID' values for specific devices:

Device type ID	Device description
22	Beacon HW V4.5
23	Beacon HW V4.5 (hedgehog mode)
24	Modem HW V4.9
30	Beacon HW V4.9
31	Beacon HW V4.9 (hedgehog mode)
32	Beacon Mini-RX
36	Beacon Mini-TX
37	Beacon-TX-IP67
41	Beacon industrial-RX
42	Super-Beacon
43	Super-Beacon (hedgehog mode)
44	Industrial Super-Beacon
45	Industrial Super-Beacon (hedgehog mode)
46	Super-Modem
48	Modem HW V5.1

You can get device type id from [devices list](#) and [reading device version](#) commands.

## 4. Protocols of communication via RS-485

### 4.1 'Marvelmind' protocol for streaming

#### Supported hardware:

Super-Beacon:	not supported
Industrial Super-Beacon:	supported
Modem HW5.1:	not supported
Super-Modem:	supported
Mini-Rx (Badge, Helmet, etc.):	not supported
Mini-Tx:	not supported
Modem HW4.9:	not supported
Beacon HW4.9:	not supported
Beacon HW4.5:	not supported

All packets described in [corresponding section](#) for UART are also available via RS-485.

Note these data are available only for Super-Modem and Industrial Super-Beacon, because they have RS-485 hardware onboard.

## 4.2 Protocol of reading/writing data from/to user device

### Supported hardware:

Super-Beacon:	not supported
Industrial Super-Beacon:	on demand
Modem HW5.1:	not supported
Super-Modem:	on demand
Mini-Rx (Badge, Helmet, etc.):	not supported
Mini-Tx:	not supported
Modem HW4.9:	not supported
Beacon HW4.9:	not supported
Beacon HW4.5:	not supported

All packets described in [corresponding section](#) for UART can be implemented on demand.

Note these data can be available only for Super-Modem and Industrial Super-Beacon, because they have RS-485 hardware onboard.

## 4.3 NMEA0183 communication protocol

### Supported hardware:

Super-Beacon:	not supported
Industrial Super-Beacon:	supported
Modem HW5.1:	not supported
Super-Modem:	supported (starting from SW V7.000)
Mini-Rx (Badge, Helmet, etc.):	not supported
Mini-Tx:	not supported
Modem HW4.9:	not supported
Beacon HW4.9:	not supported
Beacon HW4.5:	not supported

All packets described in [corresponding section](#) for UART are also available via RS-485.

Note these data are available only for Super-Modem and Industrial Super-Beacon, because they have RS-485 hardware onboard.

## 5. Protocols of communication via SPI

### 5.1 Packet with hedgehog location

#### Supported hardware:

Super-Beacon:	supported
Industrial Super-Beacon:	not supported
Modem HW5.1:	on demand
Super-Modem:	not supported
Mini-Rx (Badge, Helmet, etc.):	not supported
Mini-Tx:	not supported
Modem HW4.9:	not supported
Beacon HW4.9:	supported
Beacon HW4.5:	supported

Super-Beacon, Beacon HW4.9 and Beacon HW4.5 can work as SPI slave devices and support reading packet with [hedgehog location data](#). Modem HW5.1 has hardware SPI support and software support can be added on demand.

## 5.2 Other data via SPI

### Supported hardware:

Super-Beacon:	on demand
Industrial Super-Beacon:	not supported
Modem HW5.1:	on demand
Super-Modem:	not supported
Mini-Rx (Badge, Helmet, etc.):	not supported
Mini-Tx:	not supported
Modem HW4.9:	not supported
Beacon HW4.9:	not supported
Beacon HW4.5:	not supported

Support of other data packets described in chapter 2 can be added on demand for Super-Beacon and modem HW5.1

## 6. Protocols of communication via I<sup>2</sup>C

### 6.1 Compass emulation for drones with PX4

#### Supported hardware:

Super-Beacon:	supported
Industrial Super-Beacon:	not supported
Modem HW5.1:	not supported
Super-Modem:	not supported
Mini-Rx (Badge, Helmet, etc.):	not supported
Mini-Tx:	not supported
Modem HW4.9:	not supported
Beacon HW4.9:	not supported
Beacon HW4.5:	not supported

Paired Super-Beacons can work as more stable and precise compass connected via I2C to PX4.

You need to purchase [MMSW0003](#) license for this.

## 6.2 Other data via I<sup>2</sup>C

### Supported hardware:

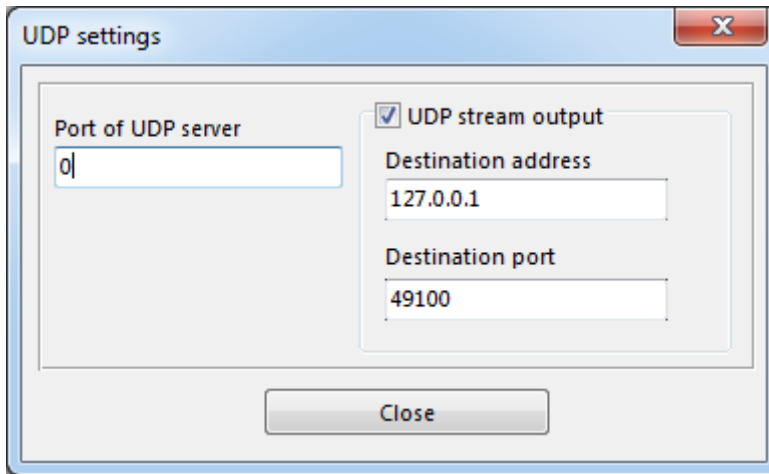
Super-Beacon:	on demand
Industrial Super-Beacon:	not supported
Modem HW5.1:	on demand
Super-Modem:	not supported
Mini-Rx (Badge, Helmet, etc.):	not supported
Mini-Tx:	not supported
Modem HW4.9:	not supported
Beacon HW4.9:	not supported
Beacon HW4.5:	not supported

Support of other data packets described in chapter 2 can be added on demand for Super-Beacon and modem HW5.1



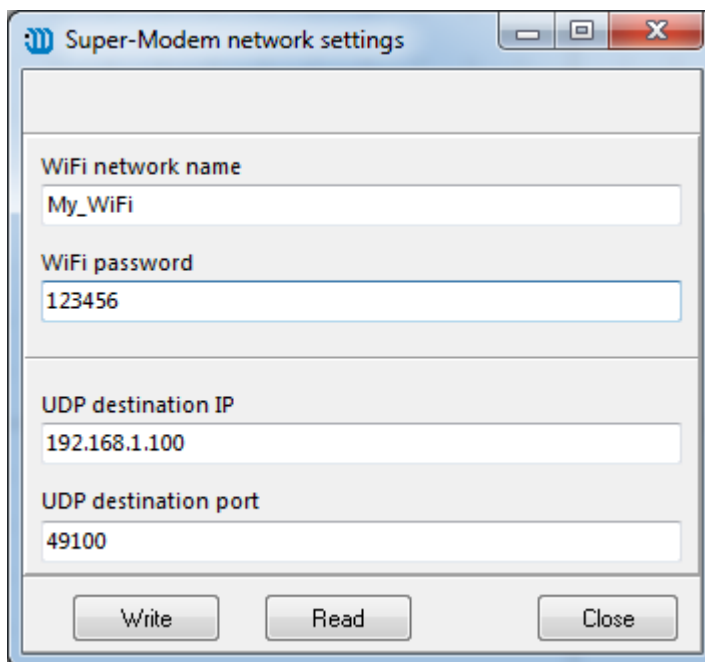
## 7. Protocols of communication via UDP (Wi-Fi)

Dashboard software can transmit data via UDP through network interfaces of the PC where the dashboard is running. Destination IP address/port can be adjusted via menu File/UDP settings:



Super-Modem has onboard Wi-Fi, and it is able to stream locations of mobile beacons.

Configuration of WiFi network and UDP streaming settings (IP address and port) is possible through dashboard – menu item 'Super-Modem network settings' from menu 'File' or from menu by right click on the Super-Modem button:



## 7.1 Packet with hedgehog location

### Supported hardware/software:

Super-Beacon:	not supported
Industrial Super-Beacon:	not supported
Modem HW5.1:	not supported
Super-Modem:	supported
Mini-Rx (Badge, Helmet, etc.):	not supported
Mini-Tx:	not supported
Modem HW4.9:	not supported
Beacon HW4.9:	not supported
Beacon HW4.5:	not supported
Dashboard software:	supported

### Format of the packet

Offset	Size	Type	Description	Value
0	1	uint8_t	Address of the beacon	
1	1	uint8_t	Type of packet	0x47
2	2	uint16_t	Code of data in packet	0x0011
4	1	uint8_t	Data size (bytes)	0x16
5	4	uint32_t	Timestamp – time from running of dashboard/Super-Modem in milliseconds on the moment of receiving coordinates	
9	4	int32_t	Coordinate X of beacon, mm	
13	4	int32_t	Coordinate Y of beacon, mm	
17	4	int32_t	Coordinate Z of beacon, mm	
21	1	uint8_t	Byte of flags: Bit 0: 1 - coordinates unavailable. Data from fields X,Y,Z should not be used. Bit 1: timestamp units indicator (see note) Bit 2...6: reserved (0) Bit 7: – 1 – out of geofencing zone	
22	1	uint8_t	Reserved (0)	
23	2	uint16_t	Bit 0...11: orientation of hedgehogs pair in XY plane, decidegrees (0...3600) Bit 12: 1 – coordinates are given for center of beacons pair; 0 – coordinates for specified hedgehog Bit 13...15: reserved (0)	
24	5	Reserved (0)		

Note: for dashboard and Super-Modem versions before V6.290 the timestamp is in 1/64 sec units and timestamp units indicator (bit 1 of flags byte) is 0. For versions V6.290 and higher timestamp is in milliseconds and timestamp units indicator is 1.

## 7.2 Packet with stationary beacons locations

### Supported hardware/software:

Super-Beacon:	not supported
Industrial Super-Beacon:	not supported
Modem HW5.1:	not supported
Super-Modem:	supported
Mini-Rx (Badge, Helmet, etc.):	not supported
Mini-Tx:	not supported
Modem HW4.9:	not supported
Beacon HW4.9:	not supported
Beacon HW4.5:	not supported
Dashboard software:	supported

### Format of the packet

Offset	Size (bytes)	Type	Description	Value
0	1	uint8_t	Address	0xff
1	1	uint8_t	Type of packet	0x47
2	2	uint16_t	Code of data in packet	0x0012
4	1	uint8_t	Number of bytes of data transmitting	1+N*14
5	1	uint8_t	Number of beacons in packet	N
6	1	N*14 bytes	Data for N beacons	

### Format of data structure for every of N beacons:

Offset	Size (bytes)	Type	Description
0	1	uint8_t	Address of the beacon
1	4	int32_t	Coordinate X of the beacon, mm
5	4	int32_t	Coordinate Y of the beacon, mm
9	4	int32_t	Coordinate Z of the beacon, mm
13	1	uint8_t	Reserved (0)

## 7.3 Packet with raw IMU data

### Supported hardware/software:

Super-Beacon:	not supported
Industrial Super-Beacon:	not supported
Modem HW5.1:	not supported
Super-Modem:	supported
Mini-Rx (Badge, Helmet, etc.):	not supported
Mini-Tx:	not supported
Modem HW4.9:	not supported
Beacon HW4.9:	not supported
Beacon HW4.5:	not supported
Dashboard software:	supported

### Format of the packet

Offset	Size (bytes)	Type	Description	Value
0	1	uint8_t	Address of the beacon	
1	1	uint8_t	Type of packet	0x47
2	2	uint16_t	Code of data in packet	0x0003
4	1	uint8_t	Number of bytes of data transmitting	0x20
5	2	int16_t	Accelerometer, X axis, 1 mg/LSB	
7	2	int16_t	Accelerometer, Y axis, 1 mg/LSB	
9	2	int16_t	Accelerometer, Z axis, 1 mg/LSB	
11	2	int16_t	Gyroscope, X axis, 0.0175 dps/LSB	
13	2	int16_t	Gyroscope, Y axis, 0.0175 dps/LSB	
15	2	int16_t	Gyroscope, Z axis, 0.0175 dps/LSB	
17	2	int16_t	Compass, X axis, 1100 LSB/Gauss	
19	2	int16_t	Compass, Y axis, 1100 LSB/Gauss	
21	2	int16_t	Compass, Z axis, 980 LSB/Gauss	
23	1	uint8_t	Address of the beacon	
24	5	5 bytes	Reserved (0)	
29	4	uint32_t	Timestamp, ms	
33	8	8 bytes	reserved	

Note: Compass data are available only for HW v4.9 beacons with IMU.

## 7.4 Packet with raw distances data

### Supported hardware/software:

Super-Beacon:	not supported
Industrial Super-Beacon:	not supported
Modem HW5.1:	not supported
Super-Modem:	supported
Mini-Rx (Badge, Helmet, etc.):	not supported
Mini-Tx:	not supported
Modem HW4.9:	not supported
Beacon HW4.9:	not supported
Beacon HW4.5:	not supported
Dashboard software:	supported

### Format of the packet

Offset	Size (bytes)	Type	Description	Value
0	1	uint8_t	Address	0xff
1	1	uint8_t	Type of packet	0x47
2	2	uint16_t	Code of data in packet	0x0004
4	1	uint8_t	Number of bytes of data transmitting	0x20
5	32		Data packet (see lower)	

### Format of data packet

Offset	Size (bytes)	Type	Description	Value
0	1	uint8_t	Address of hedgehog	
1	6		Distance item 1	
7	6		Distance item 2	
13	6		Distance item 3	
19	6		Distance item 4	
25	4	uint32_t	Timestamp – internal time of beacon ultrasound emission, in milliseconds from the moment of the latest wakeup event (V5.89+).	
29	2	uint16_t	Time passed from ultrasound emission to current time, milliseconds (V5.89+)	
31	1	uint8_t	reserved	

### Format of distance item

Offset	Size (bytes)	Type	Description	Value
0	1	uint8_t	Address of beacon (0 if item not filled)	
1	4	uint32_t	Distance to the beacon, mm	
5	1	uint8_t	Reserved (0)	

## 7.5 Packet with IMU fusion data

### Supported hardware/software:

Super-Beacon:	not supported
Industrial Super-Beacon:	not supported
Modem HW5.1:	not supported
Super-Modem:	supported
Mini-Rx (Badge, Helmet, etc.):	not supported
Mini-Tx:	not supported
Modem HW4.9:	not supported
Beacon HW4.9:	not supported
Beacon HW4.5:	not supported
Dashboard software:	supported

### Format of the packet

Offset	Size (bytes)	Type	Description	Value
0	1	uint8_t	Address of the beacon	
1	1	uint8_t	Type of packet	0x47
2	2	uint16_t	Code of data in packet	0x0005
4	1	uint8_t	Number of bytes of data transmitting	0x2a
5	4	int32_t	Coordinate X of beacon (fusion), mm	
9	4	int32_t	Coordinate Y of beacon (fusion), mm	
13	4	int32_t	Coordinate Z of beacon (fusion), mm	
17	2	int16_t	W field of rotation quaternion	
19	2	int16_t	X field of rotation quaternion	
21	2	int16_t	Y field of rotation quaternion	
23	2	int16_t	Z field of rotation quaternion	
25	2	int16_t	Velocity X of beacon (fusion), mm/s	
27	2	int16_t	Velocity Y of beacon (fusion), mm/s	
29	2	int16_t	Velocity Z of beacon (fusion), mm/s	
31	2	int16_t	Acceleration X of beacon, mm/s <sup>2</sup>	
33	2	int16_t	Acceleration Y of beacon, mm/s <sup>2</sup>	
35	2	int16_t	Acceleration Z of beacon, mm/s <sup>2</sup>	
37	1	uint8_t	Address of beacon	
38	1	1 byte	Reserved (0)	
39	4	uint32_t	Timestamp, ms	
43	4	4 bytes	Reserved (0)	

Note: Quaternion is normalized to 10000 value.

## 7.6 Packet with telemetry data

### Supported hardware/software:

Super-Beacon:	not supported
Industrial Super-Beacon:	not supported
Modem HW5.1:	not supported
Super-Modem:	supported
Mini-Rx (Badge, Helmet, etc.):	not supported
Mini-Tx:	not supported
Modem HW4.9:	not supported
Beacon HW4.9:	not supported
Beacon HW4.5:	not supported
Dashboard software:	supported

### Format of the packet

Offset	Size (bytes)	Type	Description	Value
0	1	uint8_t	Address of the beacon	
1	1	uint8_t	Type of packet	0x47
2	2	uint16_t	Code of data in packet	0x0006
4	1	uint8_t	Number of bytes of data transmitting	0x10
5	2	uint16_t	Battery voltage, mV	
7	1	int8_t	RSSI, dBm	
8	13		Reserved (0)	

## 7.7 Packet with quality and extended location data

### Supported hardware/software:

Super-Beacon:	not supported
Industrial Super-Beacon:	not supported
Modem HW5.1:	not supported
Super-Modem:	supported
Mini-Rx (Badge, Helmet, etc.):	not supported
Mini-Tx:	not supported
Modem HW4.9:	not supported
Beacon HW4.9:	not supported
Beacon HW4.5:	not supported
Dashboard software:	supported

### Format of the packet

Offset	Size (bytes)	Type	Description	Value
0	1	uint8_t	Address	
1	1	uint8_t	Type of packet	0x47
2	2	uint16_t	Code of data in packet	0x0007
4	1	uint8_t	Number of bytes of data transmitting	0x10
5	1	uint8_t	Device address	
6	1	uint8_t	Positioning quality, %	
7	1	uint8_t	0 = no geofencing zone alarm 1...255 - index of geofencing zone	
8	13		Reserved (0)	



# 8. Protocols of communication via CAN

**Supported hardware:**

Super-Beacon:	not supported
Industrial Super-Beacon:	supported
Modem HW5.1:	not supported
Super-Modem:	on demand
Mini-Rx (Badge, Helmet, etc.):	not supported
Mini-Tx:	not supported
Modem HW4.9:	not supported
Beacon HW4.9:	not supported
Beacon HW4.5:	not supported

CAN hardware support can be installed in Super-Modem and Industrial Super-Beacon by request. If CAN is installed, RS-485 is not available.

Parameters of CAN:

Baudrate: **125 kbps.**

Frame format: **standard.**

## 8.1 'Marvelmind' protocol of streaming

Packets described in [corresponding chapter about UART streaming](#) are transmitted also via CAN with CAN frame id **0x10**. Each CAN frame can contain from 1 to 8 bytes of data. Number of data bytes is specified in **DLC** field of CAN frame.

Data are transmitted as raw stream, so CAN frame can include end of one data packet and beginning of next packet. User should receive multiple CAN frames, place their data fields into some buffer and process by the same way as data received from UART.

## 8.2 NMEA0183 communication protocol

Packets described in [corresponding chapter about UART streaming](#) are transmitted also via CAN with CAN frame id **0x11**. Each CAN frame can contain from 1 to 8 bytes of data. Number of data bytes is specified in **DLC** field of CAN frame.

Data are transmitted as raw stream, so CAN frame can include end of one data packet and beginning of next packet. User should receive multiple CAN frames, place their data fields into some buffer and process by the

## 9. Format of dashboard csv log file

Dashboard stores locations of stationary and mobile beacons and other data into csv log files located in 'log' folder in dashboard directory. Starting from version V7.000 format of the log was changed. Previous format was remained only for modem HW v4.9.

## 9.1 Format of csv log file (dashboard version V7.000+)

In the csv log file for dashboard versions V7.000+ each event is recorded to the log as one CSV line, and different events correspond to different formats of the line. At the same time, starting of the line is equal for all types of the line.

Here is the example of several lines from the csv log file:

```
T2021_11_04__173001_581,user,41,17,14,4.675,2.714,0.250,2,975,100
T2021_11_04__173001_581,user,41,17,15,4.665,2.708,0.250,2,975,114
T2021_11_04__173001_581,user,41,17,26,4.073,1.987,0.250,2,3462,128
T2021_11_04__173001_581,user,41,17,27,4.075,1.987,0.250,2,3462,141
T2021_11_04__173001_581,user,41,17,28,3.588,1.979,0.250,2,3496,155
T2021_11_04__173001_581,user,41,17,29,3.592,1.978,0.250,2,3496,169
T2021_11_04__173001_701,user,43,15,nl
T2021_11_04__173001_728,user,43,27,nl
T2021_11_04__173001_756,user,43,29,nl
```

Common part of the line includes first 3 fields:

- “T2021\_11\_04\_\_173001\_581” – timestamp for data from this line: 2021.11.04, 17:30:01.581;
- “user” – user name (reserved for future\_. In future versions dashboard will support logging in users;
- “41” – ID of the line type. Different line types have different formats in following fields.

There are some common special codes in data fields:

- “nl” – no license. Some license is required for this field to be filled;
- “na” – not applicable. No relevant data for this field. For example if mobile beacon was not successfully located, fields for X,Y,Z coordinates will contain “na”.

Next chapters contain descriptions of different types of the lines.

### 9.1.1 Line type ID 01 – link to map file

This line is recorded when map file is saved automatically or by 'Save map' button pressed by user.

Fields of the line:

N	Field description
0	Timestamp (common field)
1	User name (common field)
2	01 - Line type ID (link to map file)
3	Name of the map file saved at that moment

### 9.1.2 Line type ID 41 – Marvelmind protocol streaming record

This line is recorded when modem protocol setting in 'Interfaces' section is 'Marvelmind'.

[Marvelmind protocol](#) has different types of records, and they correspond to different lines in log file, described in following sub chapters.

#### 9.1.2.1 Hedgehog position (41 17)

Fields of the line:

N	Field description
0	Timestamp (common field)
1	User name (common field)
2	41 - Line type ID (Marvelmind protocol streaming)
3	17 (0x0011) – data code for hedgehog position
4	Hedgehog address
5	Hedgehog X coordinate, meters
6	Hedgehog Y coordinate, meters
7	Hedgehog Z coordinate, meters
8	Flags: Bit 0: 1 - coordinates unavailable. Data from fields X,Y,Z should not be used. Bit 1...6: reserved Bit 7: – 1 – out of geofencing zone
9	Yaw angle and flags: Bit 0...11: yaw angle of hedgehogs pair, decidegrees (0...3600) Bit 12: 1 – coordinates are given for center of beacons pair; 0 – coordinates for specified hedgehog
10	Time shift, ms. Time passed from ultrasound emission to calculation of the location in this line

### 9.1.2.2 Stationary beacon position (41 18)

Fields of the line:

N	Field description
0	Timestamp (common field)
1	User name (common field)
2	41 - Line type ID (Marvelmind protocol streaming)
3	18 (0x0012) – data code for stationary beacon position
4	Stationary beacon address
5	Beacon X coordinate, meters
6	Beacon Y coordinate, meters
7	Beacon Z coordinate, meters
8	Reserved field

### 9.1.2.3 Raw distances from hedgehog to stationary beacons (41 4)

Fields of the line:

N	Field description
0	Timestamp (common field)
1	User name (common field)
2	41 - Line type ID (Marvelmind protocol streaming)
3	4 (0x0004) – data code for raw distances position
4	Address of hedgehog
5	N – number of distances in the line
6	N distance sub records (2*N fields), see below
6+N*2+1	Time shift, ms. Time passed from ultrasound emission to measurement of the distances

Fields of the distance sub record:

0	Address of stationary beacon
1	Distance to stationary beacon



#### 9.1.2.4 Raw IMU data (41 3)

This line requires [SW Pack v7.1xx](#) license.

Fields of the line:

N	Field description
0	Timestamp (common field)
1	User name (common field)
2	41 - Line type ID (Marvelmind protocol streaming)
3	3 (0x0003) – data code for raw IMU data
4	Address of hedgehog
5	Accelerometer, X axis, 1 mg/LSB
6	Accelerometer, Y axis, 1 mg/LSB
7	Accelerometer, Z axis, 1 mg/LSB
8	Gyroscope, X axis, 0.0175 dps/LSB
9	Gyroscope, Y axis, 0.0175 dps/LSB
10	Gyroscope, Z axis, 0.0175 dps/LSB
11	Compass, X axis, 1100 LSB/Gauss
12	Compass, Y axis, 1100 LSB/Gauss
13	Compass, Z axis, 980 LSB/Gauss

### 9.1.2.5 IMU fusion data (41 5)

This line requires [SW Pack v7.1xx](#) license.

Fields of the line:

N	Field description
0	Timestamp (common field)
1	User name (common field)
2	41 - Line type ID (Marvelmind protocol streaming)
3	5 (0x0005) – data code for IMU fusion data
4	Address of hedgehog
5	Coordinate X of beacon (fusion), meters
6	Coordinate Y of beacon (fusion), meters
7	Coordinate Z of beacon (fusion), meters
8	W field of rotation quaternion
9	X field of rotation quaternion
10	Y field of rotation quaternion
11	Z field of rotation quaternion
12	Velocity X of beacon (fusion), mm/s
13	Velocity Y of beacon (fusion), mm/s
14	Velocity Z of beacon (fusion), mm/s
15	Acceleration X of beacon, mm/s <sup>2</sup>
16	Acceleration Y of beacon, mm/s <sup>2</sup>
17	Acceleration Z of beacon, mm/s <sup>2</sup>

### 9.1.2.6 Telemetry data (41 6)

Fields of the line:

N	Field description
0	Timestamp (common field)
1	User name (common field)
2	41 - Line type ID (Marvelmind protocol streaming)
3	6 (0x0006) – data code for telemetry data
4	Address of the beacon
5	Supply voltage, V
6	RSSI, dBm

### 9.1.2.7 Quality and extended location data (41 7)

Fields of the line:

N	Field description
0	Timestamp (common field)
1	User name (common field)
2	41 - Line type ID (Marvelmind protocol streaming)
3	7 (0x0007) – data code for quality and extended location data
4	Address of the hedgehog
5	Quality of the location, %
6	Number of the geofencing zone (this field requires <a href="#">SW Pack v7.1xx</a> license)

### 9.1.3 Line type ID 42 – NMEA0183 streaming record

This line requires [SW Pack v7.1xx](#) license.

This line is recorded when modem protocol setting in 'Interfaces' section is 'NMEA0183'.

Fields of the line:

N	Field description
0	Timestamp (common field)
1	User name (common field)
2	42 - Line type ID (NMEA0183 protocol streaming)
3	Address of the hedgehog
4,5, etc	Sequence of fields according to <a href="#">NMEA0183</a> format (NMEA0183 record has also comma separated values format)

#### 9.1.4 Line type ID 43 – user payload data transmitted through the hedgehog

This line requires [SW Pack v7.1xx](#) license.

This line is recorded if hedgehog has non-zero payload data size enabled in the interfaces section of settings, and user device transmits any data via USB or UART of the hedgehog.

Also, payload data are available for some Marvlemind devices, for example robots [V100](#) and [Boxie](#).

Fields of the line:

N	Field description
0	Timestamp (common field)
1	User name (common field)
2	43 - Line type ID (user payload)
3	Address of the hedgehog
4,5, etc	Sequence of comma separated bytes of payload data (each field is 1 byte)

### 9.1.5 Line type ID 44 – dashboard real-time player location

This line requires [SW Pack v7.1xx](#) license.

This line is recorded for hedgehog if real-time player is enabled. Real-time player provides 100 Hz location data.

Fields of the line:

N	Field description
0	Timestamp (common field)
1	User name (common field)
2	44 - Line type ID (real-time player location)
3	Address of the hedgehog
4	Reserved field
5	Hedgehog X coordinate, meters
6	Hedgehog Y coordinate, meters
7	Hedgehog Z coordinate, meters

## 9.2 Previous format of csv log (dashboard before V7.000 or modem HW v4.9)

Here is the picture illustrating old format of the log file:

### Format of CSV file recorded by dashboard

```
1608733078625,0,2911360,60,0.805,1.160,1.000,50,1.784,58,1.519,255,0,0,0,0,
1608733078656,31,2911391,60,0.805,1.160,1.000,50,1.784,58,1.519,255,0,0,0,0,
1608733078656,0,2911391,60,0.805,1.160,1.000,50,1.784,58,1.519,255,0,0,0,0,
1608733078671,15,2911406,60,0.805,1.160,1.000,50,1.784,58,1.519,255,0,0,0,0,
1608733078671,0,2911406,60,0.805,1.160,1.000,50,1.784,58,1.519,255,0,0,0,0,
1608733078687,16,2911422,60,0.805,1.160,1.000,50,1.784,58,1.519,255,0,0,0,0,
1608733078687,0,2911422,60,0.805,1.160,1.000,50,1.784,58,1.519,255,0,0,0,0,
1608733078843,156,2911578,60,0.806,1.159,1.000,50,1.784,58,1.519,255,0,0,0,0,
1608733078843,0,2911578,60,0.806,1.159,1.000,50,1.784,58,1.519,255,0,0,0,0,
1608733078859,16,2911594,60,0.806,1.159,1.000,50,1.778,58,1.528,255,0,0,0,0,
1608733078859,0,2911594,60,0.806,1.159,1.000,50,1.778,58,1.528,255,0,0,0,0,
1608733078890,31,2911625,60,0.807,1.159,1.000,50,1.778,58,1.528,255,0,0,0,0,
1608733078890,0,2911625,60,0.807,1.159,1.000,50,1.778,58,1.528,255,0,0,0,0,
```

Case specific parameters: robot state, some users telemetry or so on

32-bit status word.  
Bit 0: 1 = geofencing zone alarm  
Bit 1...31: reserved

Separator. 255 means end of raw distances list

Addresses of stationary beacons and distances from hedgehog to them (meters)

Coordinates of hedgehog: X,Y,Z (meters)

Address of hedgehog

Time from running dashboard, ms

Time from previous record, ms

Time from 1 january 1970 (Unix time), ms

## 10. Contacts

For additional support, please send your questions to [info@marvelmind.com](mailto:info@marvelmind.com)



## Appendix 1. Calculating CRC-16

For checksum the CRC-16 is used. Last two bytes of N-bytes frame are filled with CRC-16, applied to first (N-2) bytes of frame. To check data, you can apply CRC-16 to all frame of N bytes, the result value should be zero.

Below is the implementation of the algorithm in the 'C':

```
typedef ushort ModbusCrc;// ushort – two bytes

typedef union {
    ushort w;
    struct{
        uchar lo;
        uchar hi;
    } b;
    ucharbs[2];
} Bytes;

static Modbus CrcmodbusCalcCrc(const void *buf, ushort length)
{
    uchar *arr = (uchar *)buf;
    Bytes crc;

    crc.w = 0xffff;

    while(length--){
        chari;
        bool odd;

        crc.b.lo ^= *arr++;
        for(i = 0; i < 8; i++){
            odd = crc.w & 0x01;
            crc.w >>= 1;
            if(odd)
                crc.w ^= 0xa001;
        }
    }
    return (ModbusCrc)crc.w;
}
```

## Appendix 2. Format of error reply from modem

**Format of error frame (from modem to host)**

Offset	Size (bytes)	Type	Description	Value
0	1	uint8_t	Address of modem	0xff
1	1	uint8_t	Type of packet	
2	1	uint8_t	Code of error	
3	2	uint16_t	CRC-16 (see appendix 1)	

Type of the error packet is the type of packet for the request frame with added high bit. For example, if type of packet for request is 0x03, the type of error packet will be 0x83.

Code of error may be one of following:

- 1 – unknown type of packet in request
- 2 – unknown code of data in request
- 3 – error in data field of request
- 6 – device is busy
- 10 – error message from remote device
- 11 – timeout of reply from remote device