Protocol of data exchange with mobile beacon via USB and UART interfaces

Version 2015.11.18
Valid for firmware v4.07 and newer

To get location 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 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, firmware versions V4.02+ support configurable from the Dashboard baudrate 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.

The hedgehog constantly streams out packets of data without any request.

Data is represented in binary format.

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

Marvelmind robotics

Packets format

1. General packet format

All packets have same general structure:

Offset	Size (bytes)	Type	Description	Value
0	1	uint8_t	Address (constant value now)	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)	

1.1. Packet of hedgehog coordinates (code of data 0x0001).

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

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	0x0001
4	1	uint8_t	Number of bytes of data transmitting	0x10
5	4	uint32_t	Timestamp – internal time of beacon measured in alpha-cycle periods (1/64 sec) from the moment of the latest wakeup event	
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…7 – reserved (0)	
16	5	5 bytes	Reserved (0)	
21	2	uint16_t	CRC-16 (see appendix)	

1.2. Packet of all beacons coordinates (code of data 0x0002).

This packet is transmitted when system becomes frozen, and repeats every 10 sec.

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)	

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)

Appendix1. 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;
}
```

Appendix2. Data dump