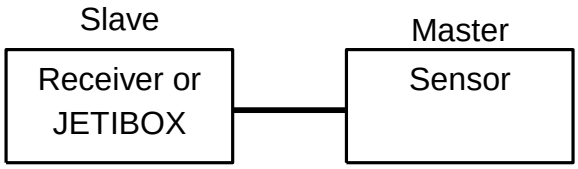
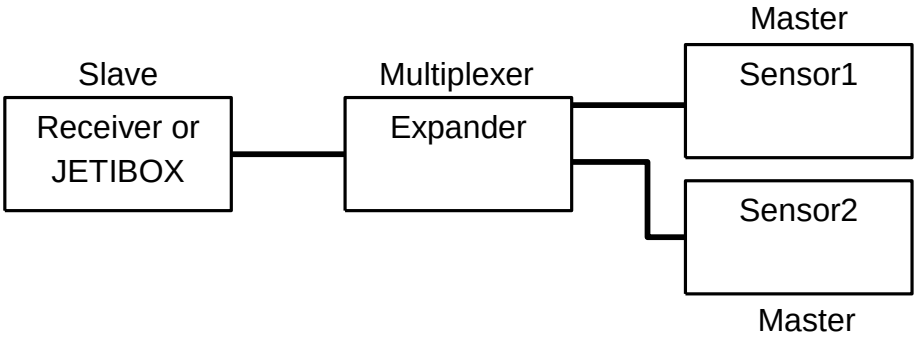
JETI Telemetry communication protocol

This document describes a telemetry communication protocol that is used by sensors of JETI model. Telemetry information is transmitted through a one-wire bus from sensors to receiver. As a following step, the information is wirelessly sent to another device, such as transmitter, TX module or JETIBOX profi.

Topology

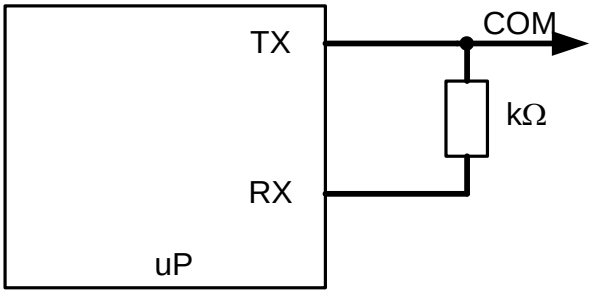
The bus has a topology called "point to point". Every sensor is marked as a "master" because it always initiates communication. A device connected to the sensor is marked as a "slave". If several sensors are connected, it is mandatory to use a multiplexer (Expander), that processes all inputs and transforms them into one single output.





Physical layer

Communication is realized by serial asynchronous interface (UART) in a half-duplex mode.
Communication speed **9600-9800 Baud**
Number of data bits **9**
Number of stop bits **2**
ODD parity
Communication lines RX and TX are physically connected through a resistor with resistance of a few kilo-ohms (see the following picture).



Logic levels:
Maximum level of logical zero "0": 1,5V
Minimum level of logical one "1": 3,0V

Accessing shared data bus

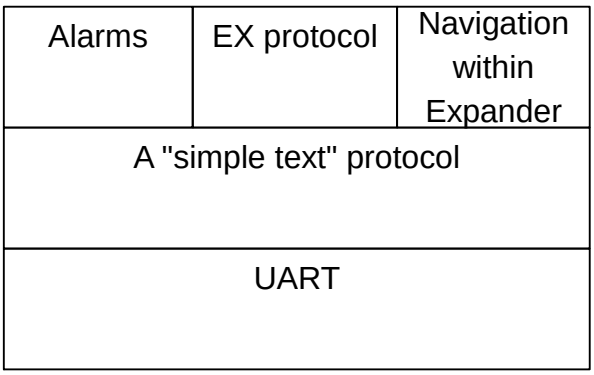
The only one active element of a network at a time, and which initiates all communications, is a "master". It sends a packet and then releases the bus for at least 20ms. At this time, "slave" is allowed to respond. This situation repeats periodically. "Slave" doesn't have to respond immediately and it doesn't have to respond at all.

After transmission, the bus is released in the following way. The TX line is reconfigured as an input with use of an additional internal pull-up resistor. After timeout has passed, the TX line returns into output state. "Slave" (JETIBOX) sends information about buttons pressed within specified time.

Position of a byte	8 data bits	Bit No 8	Note
1	0bLDUR0000	0	bit L – zero if *Left b*utton pressed, otherwise 1 bit D - zero if *Down b*utton pressed, otherwise 1 bit U - zero if *Up b*utton pressed, otherwise 1 bit R - zero if *Right b*utton pressed, otherwise 1

Network protocols

Currently there are four protocols existing. Protocol of the lowest layer is called "simple text" and is included in every communication packet. The rest of protocols are parts of a higher layer. During a procedure of packet construction, the message of higher-layer protocol is concatenated with a "simple text" protocol. Single data packet might contain up to one message of a higher-layer protocol.



Protocol of alarms

Alarm message informs about exceeding maximum/minimum values set in the devices. Protocol includes a letter encoded in Morse Code alphabet, which is later acoustically signalized.

Byte No.	8 data bits	Bit No. 8	Note
1	0x7E	0	Separator of a message
2	0xNL	1	L = number of bytes following (always 2), N can be any number.
3	0x22/0x23	1	0x22- without reminder tone (Vario); 0x23 – with reminder tone (standard alarm)
4	'A'-'Z'	1	ASCII letter to be signalized by Morse alarm

Protocol EX

The EX protocol exists in two forms – *text* and *data* protocol. *Data* specification is dedicated to transmitting sensors' values directly in a binary form. *Text* specification provides human-understandable labels to *data* specification.

For example a temperature sensor, ***SENSOR T***:

Identifier	Data specification	Text specification	Meaning
0	reserved	"SENSOR T"	Description of sensor
1	40	"Temper 1",“°C“	Temperature 1 = 4.0°C
2	80	"Temper 2",“°C“	Temperature 2 = 8.0°C

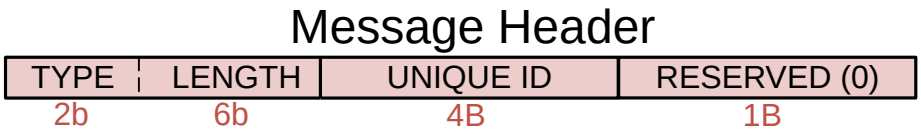
Data packet transmits information about temperature measured on input 1 and input 2. Text packets describe meaning of those data together with device description.

Protocol Header

Definite distinction of EX protocol locates in a **second byte** of a message. If the second byte has value **0xNF (for example 0x2F)**, the protocol is definitely EX. The specification of EX protocol follows. Length of the packet counts from zero starting at the **third byte (meaning the bytes following)**. Byte ordering is implemented as **Little Endian**. Serial number of a device is divided into two two-byte values.

For third party devices you are free to use the upper part of a serial number within range 0xA400 – 0xA41F. The lower part of a serial number should be used in a manner it ensures uniqueness of the whole serial number (4B). Maximum allowed length of a packet is 29B (together with separators 0x7E; 0xNF).

Byte No.	Length	8 data bits	Bit No. 8	Note
1	1B	0x7E	0	Separator of the message
2	1B	0xNF	1	Distinct identification of an EX packet, N could be an arbitrary number.
3	2b	Type (0-3)	1	Packet type; 1 – Data protocol, 0 - Text protocol
3	6b	Length (0-31)	1	Length of a packet (number of bytes following)
4-5	2B	Serial Number	1	Upper part of a serial number, Manufacturer ID (Little Endian)
6-7	2B	Serial Number	1	Lower part of a serial number, Device ID (Little Endian)
8	1B	0x00	1	Reserved



Data specification

An identifier is used to link data values together with textual description. The identifier might be a number within range **1-15**. Special case is a zero value, which enhances range of identifiers. The definition of an identifier is located in the following byte before the whole data representation (see the table).

Byte No.	Length	8 data bits	Bit No. 8	Note
9	4b	Identifier (0-15)	1	Identifier of telemetry value
9	4b	Data type (0-15)	1	Data type of telemetry value
10	xB	Data	1	Data with length according to data type
11+x	4b	Identifier (1-15)	1	Identifier of a second telemetry value
11+x	4b	Data type (0-15)	1	Data type of a second telemetry value
12+x	yB	Data	1	Data with length according to data type
13+x+y	1B	CRC8	1	Cyclic redundancy check

Data types

All data types are signed. The upper 3 bits express a sign followed by a position of decimal point.
Example: data type int6_t

Bit	Description
7	Sign of a number (zero = positive number)
5:6	Position of decimal point (special use with some data types)
4:0	Value of a number written in a direct binary form

Data type	Description	Note
0	int6_t	Data type 6b (-31 ¸31)
1	int14_t	Data type 14b (-8191 ¸8191)
2	int14_t	Reserved
3	int14_t	Reserved
4	int22_t	Data type 22b (-2097151 ¸2097151)
5	int22_t	Special data type – time and date
6	int22_t	Reserved
7	int22_t	Reserved
8	int30_t	Data type 30b (-536870911 ¸536870911)
9	int30_t	Special data type – GPS coordinates
10	int30_t	Reserved
11	int30_t	Reserved
12	int38_t	Reserved
13	int38_t	Reserved
14	int38_t	Reserved
15	int38_t	Reserved

Special data type time and date

- If the lowest bit of a decimal point equals log. 1, the data represents *date*

(decimal representation: b0-7 day, b8-15 month, b16-20 year - 2 decimals, number 2000 to be added).

- If the lowest bit of a decimal point equals log. 0, the data represents *time*

(decimal representation: b0-7 seconds, b8-15 minutes, b16-20 hours).

Special data type Coordinates

- If the lowest bit of a decimal point equals log. 1, the data represents *longitude*. According to the highest bit (30) of a decimal point it is either **West (1)** or **East (0)**.
- If the lowest bit of a decimal point equals log. 0, the data represents *latitude*. According to the highest bit (30) of a decimal point it is either **South (1)** or **North (0)**.

Textual specification

It assigns textual description to data transmitted in a binary form. It consists of parts like a *label* (temperature) and a *unit* (°C). The zero-valued identifier is used to give a name to the device itself. Length of the description and unit counts from 1.

Byte No.	Length	8 data bits	Bit No.8	Note
9	1B	Identifier (0-255)	1	Identifier of telemetry value
10	5b	Length of the description (X)	1	Length of the description in Bytes
10	3b	Length of unit's description (Y)	1	Length of the unit's description in Bytes
11	XB	Label	1	ASCII textual description of a value
11+X	YB	Label	1	ASCII textual description of a unit
11+X+Y	1B	CRC8	1	Cyclic redundancy check

Checksum
Cyclic redundancy check is controlled by a polynomial X^8 + X^2 + X + 1 and is initiated by zero value. Counting of checksum value begins at **the third byte of a message** (length of data).

```
/* 8-bit CRC polynomial  X^8 + X^2 + X + 1 */.
#define POLY 0x07

unsigned char update_crc (unsigned char crc, unsigned char crc_seed)
{
    unsigned char crc_u;
    unsigned char i;

    crc_u = crc;
    crc_u ^= crc_seed;

    for (i=0; i<8; i++)
    {
       crc_u = ( crc_u & 0x80 ) ? POLY ^ ( crc_u << 1 ) : ( crc_u << 1 );
    }
    return crc_u;
}

unsigned char crc8 (unsigned char *crc, unsigned char crc_lenght)
{
  unsigned char crc_up = 0;
  unsigned char c;

  for(c=0;c < crc_lenght; c++) {
    crc_up = update_crc (crc[c], crc_up);
  }

  return crc_up;
}
```

Navigation within Expander menu
When accessing devices through Expander, a protocol exist that is dedicated to its control.

Byte No.	8 data bits	Bit No.8	Note
1	0x7E	0	Separator of a message
2	0xNL	1	L =length of the bytes following (always 1), N might be arbitrary
3	0x31	1	Code number that causes to leave menu of the sensor and return to the menu of Expander.

A "Simple text" protocol
This protocol consists of control characters (message separators) and a simple text written in ASCII form. The text is displayed on Jetibox. All separators have bit No. 8 set to log. zero. Other characters (simple text) have this bit set to log. 1. It is mandatory to transmit the whole packet consisting of 34 bytes and it is not possible to send only a part of text.

Byte No.	8 data bits	Bit No. 8	Note
1	0xFE	0	Separator of a message (begin)
2	'T' (0x54)	1	ASCII character
3	'E' (0x45)	1	ASCII character
4	'X' (0x58)	1	ASCII character
5	'T' (0x54)	1	ASCII character
….	….	….	….
….	….	….	….
….	….	….	….
34	0xFF	0	Separator of a message (end)

In these examples you can find data listings of messages. The parity and bits No. 8 are not included.

Examples of protocols – EX data specification

0x7E 0x9F 0x4C 0xA1 0xA8 0x5D 0x55 0x00 0x11 0xE8 0x23 0x21 0x1B 0x00 0xF4 0xFE 0x20 0x20 0x20 0x2A 0x4D 0x53 0x50 0x45 0x45 0x44 0x20 0x20 0x20 0x6D 0x2F 0x73 0x20 0x20 0x03E 0x3E 0x3E 0x3E 0x3E 0x3E 0x3E 0x3E 0x20 0x31 0x30 0x30 0x2E 0x30 0xFF

Protocol EX- *Data*
0x7E 0x9F 0x4C 0xA1 0xA8 0x5D 0x55 0x00 0x11 0xE8 0x23 0x21 0x1B 0x00 0xF4

0x7E; 0x9F - separators
0x4C – Message type = "Data" (1), length =12B (0x4C = 01:1001100b)
0xA8A1 – Upper part of a serial number (Product ID)
0x555D – Lower part of a serial number (Device ID)
0x00 – Reserved
0x11 – Identifier (1) which is represented by an **int14_t (1)** data type (0x11=0001:0001b)
0x23E8 – Position of decimal point equals to 1, raw value is 1000 = 100,0 (0x3E8)
0x21 – Identifier (2) which is represented by an **int14_t (1)** data type (0x21=0010:0001b)
0x001B – Position of decimal point equals to 0, raw value is 27 = 27 (0x1B)
0xF4 - CRC8

Protocol "simple text"
0xFE 0x20 0x20 0x20 0x2A 0x4D 0x53 0x50 0x45 0x45 0x44 0x20 0x20 0x20 0x6D 0x2F 0x73 0x20 0x20 0x03E 0x3E 0x3E 0x3E 0x3E 0x3E 0x3E 0x3E 0x20 0x31 0x30 0x30 0x2E 0x30 0xFF

0xFE – Separator of a message (begin)
32B ASCII DATA ->" *MSPEED m/s >>>>>>>> 100.0 "
0xFF – Separator of a message (end)

Examples of protocols – EX text specification

0x7E 0x9F 0x0F 0xA1 0xA8 0x5D 0x55 0x00 0x02 0x2A 0x54 0x65 0x6D 0x70 0x2E 0xB0 0x43 0x28 0xFE 0x20 0x20 0x20 0x2A 0x4D 0x53 0x50 0x45 0x45 0x44 0x20 0x20 0x20 0x6D 0x2F 0x73 0x20 0x20 0x3E 0x3E 0x3E 0x3E 0x3E 0x3E 0x3E 0x3E 0x20 0x31 0x30 0x30 0x2E 0x30 0xFF

Protocol EX- Text
0x7E 0x9F 0x0F 0xA1 0xA8 0x5D 0x55 0x00 0x02 0x2A 0x54 0x65 0x6D 0x70 0x2E 0xB0 0x43 0x28

0x7E; 0x9F – separators
0x0F – Message type Text (0), length = 15B
0xA8A1 – Upper part of a serial number (Product ID)
0x555D – Lower part of a serial number (Device ID)
0x00 – Reserved
0x02 – Identifier (2)
0x2A – Length of description (5), length of unit's label (2) (0x2A=0010:1010b)
0x54 0x65 0x6D 0x70 x02E - ASCII „Temp." (compatible with HD44780)
0xB0 0x43 - ASCII „°C" (compatible with HD44780)
0x28 - CRC

Protocol "simple text"
0xFE 0x20 0x20 0x20 0x2A 0x4D 0x53 0x50 0x45 0x45 0x44 0x20 0x20 0x20 0x6D 0x2F 0x73 0x20 0x20 0x03E 0x3E 0x3E 0x3E 0x3E 0x3E 0x3E 0x3E 0x20 0x31 0x30 0x30 0x2E 0x30 0xFF

0xFE - Separator of a message (begin)
32B ASCII DATA ->" *MSPEED m/s >>>>>>>> 100.0 "
0xFF - Separator of a message (end)

Examples of protocols - alarms

0x7E 0x92 0x23 0x59 0xFE 0x20 0x20 0x20 0x2A 0x4D 0x53 0x50 0x45 0x45 0x44 0x20 0x20 0x20 0x6D 0x2F 0x73 0x20 0x20 0x03E 0x3E 0x3E 0x3E 0x3E 0x3E 0x3E 0x3E 0x20 0x31 0x30 0x30 0x2E 0x30 0xFF

Protocol of Alarm
0x7E 0x92 0x23 0x59

0x7E – separators
0x92 – length (2)
0x23 – acoustic signalization with reminder tone
0x59 - ASCII character„Y“ -> to be indicated by Morse alarm

Protocol “simple text”
0xFE 0x20 0x20 0x20 0x2A 0x4D 0x53 0x50 0x45 0x45 0x44 0x20 0x20 0x20 0x6D 0x2F 0x73 0x20 0x20 0x03E 0x3E 0x3E 0x3E 0x3E 0x3E 0x3E 0x3E 0x20 0x31 0x30 0x30 0x2E 0x30 0xFF

0xFE - Separator of a message (begin)
32B ASCII DATA ->“ *MSPEED m/s >>>>>>>> 100.0 “
0xFF - Separator of a message (end)

You can use the communication protocol with these devices:

Device	Support since firmware version
Jetibox profi	1.17
DC-16	1.05
DS-16	