Six Transmitter Data communication Manual

If you are using our evaluation software “bioMON” this document is not relevant for you.
You may download “bioMON” as a Windows installer package at www.jobst-technologies.com/download/software/bioMON4_Setup.zip and get advice for installation and operation watching the video tutorials available at www.jobst-technologies.com/support-2/biomon-video-tutorials/.

Jobst Technologies GmbH
Freiburg, Germany
www.jobst-technologies.com
info@jobst-technologies.com
Fon: +49 761 5577520
Fax: +49 761 5577522

Six transmitter data communication

You will have either the regular version with an USB cable or the OEM version without housing and cable.

USB version

Win 10

Follow these steps:

1. Assure you have Internet access.
2. Plug in the USB cable to the PC.
3. Download our evaluation software “bioMON”:
   https://www.jobst-technologies.com/download/software/bioMON4_Setup.zip
4. Install bioMON (this requires Admin privileges). Windows 10 will then automatically install a serial-to-USB driver for SIX.
5. You can keep bioMON installed, no matter if you will use it or not.

Win 7

Follow these steps:

1. Download the serial-to-USB driver from:
   http://www.prolific.com.tw/US/ShowProduct.aspx?p_id=225&pcid=41
   that provides you a virtual COM port.
2. Plug in the USB cable to the PC.

OEM version

For the OEM version connect ground and supply voltage (3.3 – 5.5 V) to your power supply. Connect the Read data (Rx) pin of your microprocessor to the Send data (Tx) pin of the Six transmitter. The pin assignment can be found in the instrumentation datasheet *Biosensor Transmitter OEM*:
https://www.jobst-technologies.com/download/datasheets/instrumentation-datasheets/PDD0113-1_Transmitter_OEM_datasheet_en.pdf

There is no need to connect the Rx pin of the Six transmitter as the data is sent by the transmitter automatically every 1.7 seconds.

For the virtual serial port via USB interface or the TTL serial port (UART) with OEM version use port settings of 9600 baud, no parity, 8 data bits, and 1 stop bit (9600,n,8,1).

Data conversion

Having opened the port and powered up the Six transmitter it will send the data telegrams that you need to convert according to the following pages into the reading of the six sensor channels and the temperature.

You should have received factory calibration data from us as Definition text files like "216 1C.def". Its content shows like this (line numbers 1 – 7 added here for description):

1. .1526,**.278,.123**,.1526,**.284,.119**,1,1,100,100,100,100,100,100,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1
2. 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
3. -1,1,1,-1,1,1,18,0,17,17,17,17,17,17,18,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
4. **"Blank1","Glucose1","Lactate1","Blank 2","Glucose 2","Lactate 2","Temperature"**
5. 0,**1**,**1**,0,**4**,**4**,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
6. 1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1
7. 0,**3.8**,**3.2**,0,**3.8**,**3.2**,0,**32**,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0

Comments:

Line 1 holds the gain factors. Here: G_Blank1 = 0.1526, G_Glucose1 = 0.278, G_Lactate1 = 0.123, G_Blank2 = 0.1526, G_Glucose2 = 0.284, G_Lactate2 = 0.119

Line 4 holds the configuration of the chip you are using. In the example electrode 1 is **blank sensor**, etc.

Line 5 the channels to subtract. For chn1 none. For chn2 use chn1, etc.

Line 1 holds the actual calibration data. First entry 0.1526 converts chn1 readings into nA while the second entry is used to convert the difference of the AD counts of chn2-chn1 into mM of glucose, etc.

Line 7 holds the temperature sensitivity factors (e.g. **3.8 %/degC** for chn2 = glucose 1) and the reference temperature (here **32** degC).

So using a 50nA range Six the concentrations in mM calculate as:

- Glucose_Signal_1 [mM] = (Chn2-Chn1) * G_Glucose1 / 100 / exp(0.038 * (Temperature – 32))
- Lactate_Signal_1 [mM] = (Chn3-Chn1) * G_Lactate1 / 100 / exp(0.032 * (Temperature – 32))
- Glucose_Signal_2 [mM] = (Chn5-Chn4) * G_Glucose2 / 100 / exp(0.038 * (Temperature – 32))
- Lactate_Signal_2 [mM] = (Chn6-Chn4) * G_Lactate2 / 100 / exp(0.032 * (Temperature – 32))

While using a 25nA range Six (info on range found on the backside label) use all (Gain factors/2) yielding:

- Glucose_Signal_1 [mM] = (Chn2-Chn1) * (G_Glucose1 / 2) / 100 / exp(0.038 * (Temperature – 32))

etc.

Jobst Technologies GmbH
Freiburg, Germany
www.jobst-technologies.com
info@jobst-technologies.com
Fon: +49 761 5577520
Fax: +49 761 5577522

Serial Protocol

Data Telegram format

byte#	hex	Description	Comments
1.	68	Start	
2.	nn	Length	The length byte is the length from the number of data bytes + 1 (the message type id byte). Decimal value is 19.
3.	nn	Length	
4.	68	Start	
5.	04	Message type ID	Message type id is a unique number identifying the which message should be used to interpret the frame. Value is 4.
6.	xx	Channel1 Hi Byte	Frame data. If the value of one of the six channels is above 2^15 counts this represents a negative number. Then subtract 2^16. That gives a reading range of (-2^15) to +((2^15)-1) counts. That range is equivalent to either +-25nA or +-50nA (check your Six transmitters label). If the values are exactly 32767 or -32768 this indicates a value above/below the measurement boundaries i.e. an error.
7.	xx	Channel1 Lo Byte	
8.	xx	Channel2 Hi Byte	
9.	xx	Channel2 Lo Byte	
10.	xx	Channel3 Hi Byte	
11.	xx	Channel3 Lo Byte	
12.	xx	Channel4 Hi Byte	
13.	xx	Channel4 Lo Byte	
14.	xx	Channel5 Hi Byte	
15.	xx	Channel5 Lo Byte	
16.	xx	Channel6 Hi Byte	
17.	xx	Channel6 Lo Byte	
18.	xx	Temperature Hi Byte	Divide the value by 16 to get the temperature in deg C.
19.	xx	Temperature Lo Byte	
20.	xx	ID Highest Byte (MSB)	
21.	xx	ID Byte 2	
22.	xx	ID Byte 3	
23.	xx	ID Lowest Byte (LSB)	
24.	xx	Checksum	The checksum is the last byte of sum of the values of the bytes of the message type id and the frame data (Bytes 5-23).
25.	16	Stop byte	

Error Telegram format

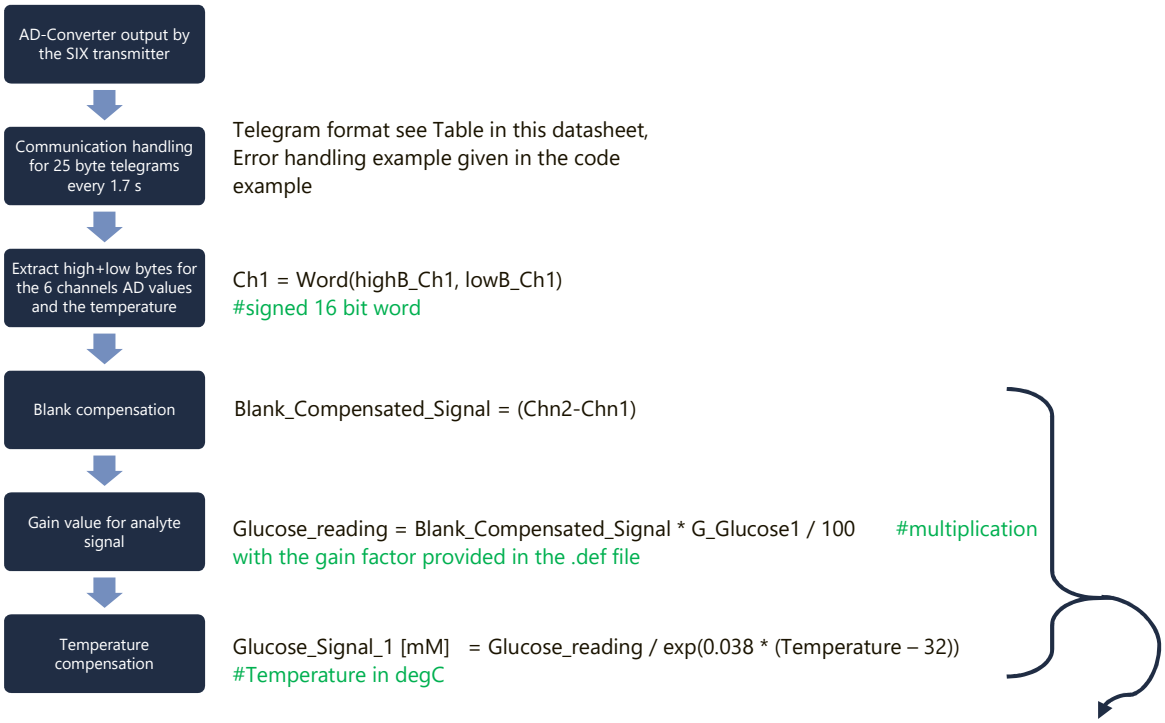
1.	68	Start	
2.	nn	Length	The length byte is the length from the number of data bytes + 1 (the message type id byte). Decimal value is 2.
3.	nn	Length	
4.	68	Start	
5.	05	Message type ID	Value for Error message is 5.
6.	xx	Error code	
7.	xx	Checksum	The checksum is the last byte of sum of the values of the bytes of the message type id and the frame data (Bytes 5-6).
8.	16	Stop byte	

Jobst Technologies GmbH
Freiburg, Germany

www.jobst-technologies.com
info@jobst-technologies.com

Fon: +49 761 5577520
Fax: +49 761 5577522

Application example



For mM reading: Glucose_Signal_1 [mM] = (Chn2-Chn1) * G_Glucose1 / 100 / exp(0.038 * (Temperature – 32))

For nA reading: Glucose_Signal_1 [nA] = (Chn2-Chn1) * G_Blank1 / 100 / exp(0.038 * (Temperature – 32))

#G_Blank1 is the blank gain factor provided in the .def file: G_Blank1 = 50/((2^15)-1)*100 = 0.1526

Example Pseudo Code on python basis

The basic code example receives the byte array and gives the current in nA (for a 50 nA transmitter) per channel as well as the temperature and a timestamp

```
# -*- coding: utf-8 -*-
For Python3.7
@author: Jobst Technologies GmbH

This simple script reads data from a COM port, to which a SIX Potentiostat is
connected. Then it stores the timestamp in s, the current in nA and the temperature in de-
gree C in a tab-delimited txt file.

The SIX potentiotat delivers a data point every 1.7 seconds, which is shown on
screen and stored a tab-delimited txt file:
    Timestamp   in s
    ch1 current in nA
    ch2 current in nA
    ch3 current in nA
    ch4 current in nA
    ch5 current in nA
    ch6 current in nA
    Temperature in °C

Start conditions:
- SIX potentiostat is connected to a known COM port
- output file name
"""
import serial
import time

# Input data:
COM_PORT = "COM5"               # This needs to be changed accordingly
OUT_FILENAME = "out_data.txt"   # This needs to be changed accordingly

# Serial port settings
BAUD = 9600
TIMEOUT = 0.5

# Data telegram settings
package_length = 25  # the SIX potentiostat delivers 25-bytes messages
data_block = [b'\x00'] * package_length  # ring array for incomming data
start_timestamp = None

# Starting serial port connection
with serial.Serial(COM_PORT, baudrate=BAUD, timeout=TIMEOUT) as ser, \
        open(OUT_FILENAME, 'w') as file:
    # output first line
    first_line = "Time/s\tCh1/nA\tCh2/nA\tCh3/nA\tCh4/nA\tCh5/nA\tCh6/nA\tT/°C"
    print(first_line)
    file.write(first_line + "\n")
    while(1):  # Script will run until Ctrl+Z or Ctrl+C is pressed
        # Read data from serial port if available in the input buffer
        data = None
        if ser.inWaiting():  # is data in input buffer?
            data = ser.read()
```

```
51.         if data is not None:
52.             # operating with a ring array of size 25.
53.             # data is inserted at 0 and removed at 25. Therefore,
54.             # the data is inverted order in the array.
55.             # array ==> [newest_byte .... oldest_byte]
56.             # Each time the code parses the ring array is parsed until
57.             # a valid telegram is recognized.
58.             data_block.insert(0, data)  # last byte in position 0
59.             nirvana = data_block.pop()
60.             del(nirvana)                # position 26 is deleted
61.             header = [b'\x04',          # Telegram header (inverted)
62.                       b'\x68',          # see documentation
63.                       b'\x13',
64.                       b'\x13',
65.                       b'\x68']
66.             cks = 0  # checksum
67.             # calculating checksum from byte 4 till second to last byte
68.             for x in [int.from_bytes(x, 'big') for x in data_block[2:-4]]:
69.                 cks = (cks + x) & 0xFF
70.             # validating header, end byte (x16) and checksum
71.             if data_block[-5:] == header \
72.                     and data_block[0] == b'\x16' \
73.                     and int.from_bytes(data_block[1], 'big') == cks:
74.                 # now inverting data train
75.                 # Useful data are from byte 5 until second to last byte.
76.                 data_inv = [x for x in data_block[2:-5]]
77.                 data_inv.reverse()
78.                 it = iter(data_inv)  # iterator used to fetch 2 bytes
79.                 # next line turns 2 bytes into a 16-bit integer array
80.                 out_data = [
81.                     int.from_bytes(b''.join([x, next(it)]),
82.                                    byteorder='big',
83.                                    signed=True) for x in it]
84.                 # data to be stored
85.                 to_save = [str(x) for x in out_data]  # 16-bit signed integers
86.                 # converting input data to currents in nanoamperes
87.                 # 50 nA ==> 32767 (2^15-1)
88.                 # applying gain and updating data as nanoamp.
89.                 gain = 50 / (2**15 - 1)
90.                 to_insert = [str(round(int(x) * gain, 3)) for x in to_save[0:6]]
91.                 # converting temperature to °C
92.                 temperature = str(round(float(to_save[6]) / 16, 3))
93.                 to_insert.append(temperature)
94.                 # generating timestamps
95.                 timestamp = round(time.time(), 4)
96.                 if start_timestamp is None:
97.                     start_timestamp = timestamp
98.                     delta_time = 0
99.                 else:
100.                     delta_time = timestamp - start_timestamp
101.                 to_insert.insert(0, str(round(delta_time, 1)))
102.                 # printing data on screen and output file
103.                 print("\t".join(to_insert))
104.                 file.write("\t".join(to_insert) + "\n")
105.                 # Next data point arrives in about 1.6 seconds
106.                 # So, resting for 1.4 seconds should be enough
107.                 time.sleep(1.4)
108.         time.sleep(0.01)   # Default polling time
```