VE.Direct Protocol FAQ

More information on the VE.Direct protocol is available here:

- Data communication whitepaper
- VE.Direct protocol document: look for VE.Direct on our whitepaper page
- Open source

Note that, besides an FAQ, this page also contains a framehandler reference implementation in C code. See at the bottom.

1. FAQ

Q1: Can I use RS232 instead of USB?

Yes, use the *VE.Direct to RS232 interface*, part number ASS030520500.

Q2: When using the VE.Direct to RS232 interface, what pins do I need?

For the communication use the GND, RX and TX pins: pin 5, 2 and 3 on the DB9 connector.

Also the DTR signal (pin 4 on the DB9 connector) and/or the RTS signal (pin 7 on the DB9 connector) must be driven high to power the isolated side of the the interface. How to program the DTR and RTS differs between used operating systems and hardware. Please note that most RS232 drivers are inverting so the logic level of the DTR must be programmed to zero in most cases. When one of those pins is not driven high, you will not be able to receive data!

Background: the “VE.Direct to RS232 interface - ASS030520500” provides galvanic isolation between the VE.Direct product and the host (your computer/PLC/etc). The VE.Direct side of the PCB is powered from VE.Direct. And the RS232 side takes its power from the DTR and RTS pins.

When using an external power supply to power DTR or RTS, the minimum voltage is 5V, and maximum is 12V. Note that it is normally not necessary to use an external power supply for this: normal serial ports will either automatically drive DTR or RTS high. And if not that, than you can control those pins to be high from within the software.

Q3: How do I calculate the HEX checksum?

I have been able to get the data I need from the sensor (BMV-700H), as show in your examples (im quoting your last email below): Consider the following example:

```
Get Battery Capacity
:70010003E<LF>     -> Command; checksum 0x55 – 0x7 – 0x0 – 0x10 – 0x0 = 0x3E
:7001000C80076<LF> -> Response; checksum 0x55 – 0x7 – 0x0 – 0x10 – 0x0 –
```

```
0xC8 – 0x0 = 0x76
```

So, this works fine when the code of the operation is less than 55 (for instance, 0x10 and 0x00 = 0x1000), and I get the same checksums as you do. However, to ask for the SOC of the batteries, Im not being able to understand how to construct the request, since the code is 0x0FFF, which is greater than 0x55. Can you provide me with an example of a request, for the SOC of the batteries? I would be much appreciated.

Anwser: VE.Direct/VE.Hex data is encoded as little endian. The checksum needs to be a byte, therefore you need to wrap it while calculating the checksum.

So in code you can loop through the message as in the following pseudo code:

```
byte checksum = 0x55;
byte message[] = { 0x7, 0xff, 0x0f, 0x00 };
for (int i =0; i < sizeof(message); i++)
   checksum -= message[i];
```

Example using a Get Command

```
Get command:
:7<register id><flags><checksum>\n

Get Soc, register 0x0FFF
:7FF0F0040\n -> Checksum = 0x55 – 7 – 0xFF – 0xF – 0 = 0x40
```

Q4: Is the VE.Direct interface 3.3 or 5V ?

It depends on the product: some are 5v, others 3.3.

The circuits in our panels and USB interfaces work on both voltages. And they automatically adapt their TX voltage to the voltage level coming from the VE.Direct product (bmv/mppt/inverter/etc).

List of the products:

- BMV-700: 3v3
- MPPT all models: 5v
- Newer MPPTs (as now under development) will remain 5V.

Q5: How much current can I draw from the power pin in the VE.Direct port?

Max 10mA average, with max 20mA/5ms bursts.

Q6: What is the end of line character when sending a HEX message?

This question concerns the situation of your software sending a HEX message to a Victron device, such as a BMV.

As also documented in the HEX protocol documents, the end of line character is `\n` aka Line feed, aka LF, aka ASCII character 10. For some Victron products, the `\r` character might work as well, when the developer added that for his convenience. But the documented and official manner to do this is using `\n`.

Q7: How about frequency of transmitting HEX messages

How often you send a HEX message doesn't matter: there is no minimum. But beware, after receiving a HEX message, the Victron product will stop sending its usual TEXT frames for a few seconds. Therefor, if you keep sending HEX messages, you'll never receive the TEXT frame any more. Which might matter in case you are depending on data in the TEXT frame.

Q8: How do I calculate the TEXT checksum?

I need to get data from a BMV, but do I check the data integrity? The VE.Direct protocol mentions the modulo 256 sum; what is this?

Answer:
Consider the following output:

```
PID 0x203
V   26201
I   0
P   0
CE  0
SOC 1000
TTG -1
Alarm   OFF
Relay   OFF
AR  0
BMV 700
FW  0307
Checksum    �
```

And as a raw data dump:

```
00000000  0d 0a 50 49 44 09 30 78  32 30 33 0d 0a 56 09 32
|..PID.0x203..V.2|
00000010  36 32 30 31 0d 0a 49 09  30 0d 0a 50 09 30 0d 0a
|6201..I.0..P.0..|
00000020  43 45 09 30 0d 0a 53 4f  43 09 31 30 30 30 0d 0a
|CE.0..SOC.1000..|
00000030  54 54 47 09 2d 31 0d 0a  41 6c 61 72 6d 09 4f 46
|TTG.-1..Alarm.OF|
00000040  46 0d 0a 52 65 6c 61 79  09 4f 46 46 0d 0a 41 52
|F..Relay.OFF..AR|
00000050  09 30 0d 0a 42 4d 56 09  37 30 30 0d 0a 46 57 09
|.0..BMV.700..FW.|
00000060  30 33 30 37 0d 0a 43 68  65 63 6b 73 75 6d 09 d8
|0307..Checksum..|
```

The checksum needs to be calculated from the start (0d) till the end of the frame, including the checksum value from the frame (d8).

In code you can loop through the message as in the following pseudo code:

```
int checksum = 0;
char message[] = { ... };
for (int i = 0; i < message.length; i++) {
   checksum = (checksum + message[i]) & 255; /* Take modulo 256 in account
*/
}

if (checksum == 0) {
   /* Checksum is valid => process message */
   handleMessage(message);
} else {
   /* Invalid checksum => reject message */
}
```

2. Framehandler reference implementation

This the main data receival routine of our consumer stack that both handles TEXT and HEX messages.

```
#include "framehandler.h"

#include <cctype>
#include <cstring>
#include <velib/utils/ve_logger.h>

#define MODULE "VE.Frame"

// The name of the record that contains the checksum.
static constexpr char checksumTagName[] = "CHECKSUM";

VeDirectFrameHandler::VeDirectFrameHandler() :
    mStop(true),
    mState(IDLE),
    mChecksum(0),
    mTextPointer(0)
{
}

void VeDirectFrameHandler::rxData(uint8_t inbyte)
{
    if (mStop) return;
    if ( (inbyte == ':') && (mState != CHECKSUM) ) {
        mState = RECORD_HEX;
    }
    if (mState != RECORD_HEX) {
        mChecksum += inbyte;
    }
    inbyte = toupper(inbyte);

    switch(mState) {
    case IDLE:
        /* wait for \n of the start of an record */
        switch(inbyte) {
        case '\n':
            mState = RECORD_BEGIN;
            break;
        case '\r': /* Skip */
        default:
            break;
        }
        break;
    case RECORD_BEGIN:
        mTextPointer = mName;
        *mTextPointer++ = inbyte;
        mState = RECORD_NAME;
        break;
    case RECORD_NAME:
        // The record name is being received, terminated by a \t
        switch(inbyte) {
        case '\t':
            // the Checksum record indicates a EOR
            if ( mTextPointer < (mName + sizeof(mName)) ) {
                *mTextPointer = 0; /* Zero terminate */
                if (strcmp(mName, checksumTagName) == 0) {
                    mState = CHECKSUM;
                    break;
                }
            }
            mTextPointer = mValue; /* Reset value pointer */
            mState = RECORD_VALUE;
            break;
        default:
            // add byte to name, but do no overflow
            if ( mTextPointer < (mName + sizeof(mName)) )
                *mTextPointer++ = inbyte;
            break;
        }
        break;
    case RECORD_VALUE:
        // The record value is being received.  The \r indicates a new
record.
        switch(inbyte) {
        case '\n':
            // forward record, only if it could be stored completely
            if ( mTextPointer < (mValue + sizeof(mValue)) ) {
                *mTextPointer = 0; // make zero ended
                textRxEvent(mName, mValue);
            }
            mState = RECORD_BEGIN;
            break;
        case '\r': /* Skip */
            break;
        default:
            // add byte to value, but do no overflow
            if ( mTextPointer < (mValue + sizeof(mValue)) )
                *mTextPointer++ = inbyte;
            break;
        }
        break;
    case CHECKSUM:
    {
        bool valid = mChecksum == 0;
        if (!valid)
            logE(MODULE,"[CHECKSUM] Invalid frame");
        mChecksum = 0;
        mState = IDLE;
        frameEndEvent(valid);
        break;
    }
    case RECORD_HEX:
        if (hexRxEvent(inbyte)) {
            mChecksum = 0;
            mState = IDLE;
        }
        break;
    }
}
```

DISQUS

~~DISQUS~~