Management software for the Voltcraft DL-191V voltage meter/data logger

Development documentation

Lázár György

August 21, 2021

Abstract

This document summarizes the main principles of handling the logger. After that, it provides full documentation for communicating with the device, which hopefully allows anyone to recreate my program from scratch. Certain encodings and their functions are documented here. For more information and examples (using libusb), please refer to the source code.

1 Introduction

1.1 Why did I make this

My idea behind this software was, that I want to provide a usable solution for every open-source lover, who ever encounters the VDL-191V. The history behind it is, that my dad purchased this piece of hardware, and he was not able to get it working under Linux. We don't really use Windows or Mac at home, and the official software did not work with wine. So I asked my teacher, if I could do this as a homework, and he agreed to it. So initially, this was kind of a school project (that's the reason for the separate Hungarian documentation). Now, it is open for anyone.

1.2 Basic handbook for the device

Since the device comes with only really basic instructions, I decided this would be a good place for a quick summary.

Before every measurement, configuration data *has* to be written to the device, or else it won't work. After the configuration is done, the device will either start the measurement instantly, or enter into a standby mode (where it consumes power), and wait for a button press to start it. This standby state is indicated by a rarely flashing green LED. It is important to note, that the device is only able to measure between 0 and 30 volts, so mind the polarity - it cannot measure like -15 volts (I'm not sure though whether it will damage the meter or not).

There are two ways to stop the measurement. One is by downloading the data from the device (well, actually you have to read the configuration data block from the device for it to stop). The other is automatic, when the logger fills its memory, or when the maximum configured data value count is reached. However, if the measurement stops this way, the meter won't turn off, instead, it will go into a standby state, waiting for the data to be downloaded. Therefore, changing the maximum data value count is not recommended. Since the device has 64kB of memory, this can be max 32000 data points (one data point is a short int, so 2 bytes).

2 Reverse engineering the communication

I used *Wireshark* to decode the communication with the device, and the original Windows software. It took a while, but after some time, most of the things were clear. The communication is simple, but it does not always follow common sense.

2.1 Basic communication

The first two packets are USB_ CONTROL_ TRANSFERs. Altough I don't fully understand what does it do, it probably sets up some kind of communication protocol between the devices. I just copied these fully from the Windows software. After the control transfers, the devices use BULK_TRANSFERs until the end of the connection, exclusively.

2.2 Communication - 3 byte headers

After setting up the communication protocol, the next step is to state our intent for the device. For this, we have to send it 3 byte headers, depending on what we want to do. These are the following (in HEX):

Get configuration data	00 10 01
Read certain part of memory	00 x y
Send configuration data	01 40 00

For these, the device replies with a 3 byte header as well. While reading the memory, these bytes are 02 00 00. For reading the configuration, the first byte is 02, but the last 2 is the number of data recorded in a short int. After writing the configuration however, the answer is only 1 byte, and it is ff if it was succesful.

2.3 The device's memory

Like I mentioned it before, the device has 64kB memory. This memory is divided up to 4kB blocks, so there are 16 blocks. The numbering starts from 0. While reading the memory, we have to set x (see the table above) to the number of the block we'd like to read from. The y value is calculated easily:

$$y = \text{the number of bytes we}'\text{d like to request from the given block} * 64 \qquad (1)$$

Therefore, the maximum value for y is 64, since 64*64 is 4096.

The keen readers might have noticed, that the header for reading the config data follows the same syntax as the memory reading header. This is absolutely true. We want to read data from the 16th block, and we'd like to read 64 bytes from it - which is the whole configuration. It seems that the device stores it's config on the end of it's memory, after it's 64kB, which is reserved for the measurement data. Note however, that reading this makes the device behave completely differently. While we can read any memory address without any consequence, reading the configuration data's memory will stop an ongoing measurement.

2.4 Configuration settings

The configuration settings turned out to be the following:

Number of bytes	Function	Encoding/Static code
0-3 (4)	Start of config data	ce 00 00 00
4-7 (4)	Max. number of samples	int
8-11 (4)	Number of measured data	int
12-15 (4)	Period of the sampling (s) (0=0.0025s)	int
16-19 (4)	Year of the start of the measurement	int
20-23 (4)	Alarm at low voltage	unknown
24-27 (4)	Alarm at high voltage	unknown
28 (1)	Month of the start of the measurement	char
29 (1)	Day of the start of the measurement	char
30 (1)	Hour of the start of the measurement	char
31 (1)	Minute of the start of the measurement	char
32 (1)	Second of the start of the measurement	char
33 (1)	Unknown (placeholder?)	00
34 (1)	Q (scroll down for more info)	bitwise
35-50 (16)	Name of the configuration	char[]/Brutus
51 (1)	Measurement start method (button/instant)	bitwise
52-55 (4)	Alarm at low voltage	unknown
56-59 (4)	Alarm at high voltage	unknown
60-63 (4)	End of config data	ce 00 00 00