VSEW_mk4

Com Protocol

January 28 2023

Bruno Paillard

1 Introduction

The *VSEW_mk4* is a new model in the *VSEW* series that introduces an open virtual Com port communication protocol. That means that the instrument can be used on any platform that has a generic driver to support the CDC (Communication) USB class. Nowadays most platforms support that class, including Windows, Mac and Linux.

That open virtual Com port only supports a subset of the instrument's commands. All commands related to memory storage, WiFi and alerts, and the settings of the instrument cannot be applied through that interface. However, that interface allows the reading of all settings, raw acceleration signals, as well as measured RMS levels in real time.

That interface allows developers to design their own application supporting the instrument.

2 Com Port Enumeration

When the instrument is enumerated by the host PC, one of the interfaces that it presents is a virtual Com port (a CDC-Class USB device). On Windows 10 and up the generic Windows Com port driver is automatically instantiated and bound to that interface. On Windows 7 and 8, even though Microsoft provides a generic driver, the user must manually load the driver when the device is connected to the PC for the first time. After the driver is loaded, a new Com port is shown in the list of devices connected to the PC.

3 Com Port Configuration

The Com port can be configured (bit rate, number of stop bits… etc.), either using the controls in Windows *Device Manager*, or in an application by using the appropriate API functions. However such settings have **no effect** on the actual communication. They are only exposed by the interface for compatibility. At the hardware level there is no physical serial line present, and the ultimate communication speed is only determined by the throughput of the USB link. That throughput is typically around 3 Mbps when there are no other devices on the USB bus.

4 Communication Structure

Exchanges between the host PC and the instrument always follow a Master-Slave model. The host initiates an exchange using a *Command Packet*. The host may also send data following that *Command Packet*. The instrument responds either with data, or with an Ack byte if no data is to be transmitted back to the host.

In all cases after sending a command, the host PC must not send another command before the instrument sends a response back. That response may be data or may be an Ack if no data is requested by the command.

5 Endianness

Unless otherwise noted, the endianness is Little-Endian.

6 Basic Types

The following basic types may be used in this protocol:

Type Name	Description	Endianness
U8	Single byte unsigned	N/A
U16	16-bit word unsigned	Little-Endian
U32	32-bit word unsigned	Little-Endian
U64	64-bit word unsigned	Little-Endian
I8	Single byte signed	N/A
I16	16-bit word signed	Little-Endian
I32	32-bit word signed	Little-Endian
I64	64-bit word signed	Little-Endian
Sgl	32-bit word in IEEE 754 floating point format	Little-Endian
Dbl	64-bit word in IEEE 754 floating point format	Little-Endian
String	Strings are concatenations of 8-bit ASCII characters, terminated by an end-of-text (0x00) byte.	N/A

Table 1

7 Packet Structure

7.1 Command Packet

The *Command Packet* is structured as follows:

Field	Size (bytes)	Function
Command	4	The command indicates the data transmitted or operation performed. The indicated direction of transmission is host-centric. Bit 31 of the command word indicates the direction of transfer: • 0: OUT (Host to Device) • 1: IN (Device to Host)
Address	4	The function of the address field varies with the command. For many commands this number is not applicable (N/A) and can be set to any value.
Count	4	This field indicates the number of items to be transferred in the following data packet (either an IN or an OUT). How the returned bytes are interpreted is defined by the command.

		For many commands this number is not applicable N/A and can be set to any value.

Table 2

7.2 Data Packet

Data Packets are simply a concatenation of bytes. The way the bytes are interpreted is a function of the command that precedes the *Data packet*.

7.3 Acknowledge

The *Ack* is a single byte with value 0x06. The *Ack* byte is only sent back to the host if the command is a *Write*, and therefore does not require a data response from the device. When the command is a *Read*, the actual data sent back to the host serves that purpose and no *Ack* byte is sent back by the instrument.

7.4 Commands

Command	Description	Addr	Count	Bytes Returned	Data/Ack
0x80000010	*Read_RMS_Amplitude* This command retrieves the current running RMS amplitude. That is an exponentially averaged amplitude, using the time constant set for the instrument. The scale of the signal is either m/s^2 for acceleration or m/s for velocity. *Note: The values take into account the static acceleration of gravity, so in order to remove the impact of that static component on the results, the high-pass filter should be set.*	N/A	N/A	12	Data: 3x 32-bit IEEE-754 Floats representing the RMS amplitude on the X, Y and Z axes Ack: No
0x80000012	*Read_Temperature* This command retrieves the temperature.	N/A	N/A	4	Data: 32-bit IEEE-754 Float representing the temperature in degC Ack: No
0x80000013	*Read_Battery*	N/A	N/A	4	Data: 32-bit IEEE-754 Float representing

	This command retrieves the battery voltage.				the battery voltage in Volts Ack: No
0x80000020	*Read_SignalType* This command returns the signal type that is currently selected	N/A	N/A	1	Data: 1 byte representing the weighting curve: 0: Acceleration 1: Velocity Ack: No
0x80000021	*Read_FS* This command reads the current sampling frequency	N/A	N/A	2	Data: U16 representing the sampling frequency in Hz. Ack: No
0x80000022	*Read_Tau* This command reads the current time constant	N/A	N/A	4	Data: 32-bit IEEE-754 Float representing the time constant in s. Ack: No
0x80000023	*Read_HighPass* This command reads the current High-Pass filter setting. The instrument returns the cutoff frequency in Hz, followed by a single byte indicating if the filter is On or Off.	N/A	N/A	5	Data: 1 32-bit IEEE-754 Float representing the high-pass frequency in Hz. 1 byte representing the state of the filter: 0: Off 1: On Ack: No
0x80000024	*Read_LowPass* This command reads the current Low-Pass filter setting. The instrument returns the cutoff frequency in Hz, followed by a single byte	N/A	N/A	5	Data: 1 32-bit IEEE-754 Float representing the low-pass frequency in Hz.

	indicating if the filter is On or Off.				1 byte representing the state of the filter: 0: Off 1: On Ack: No
0x80000025	*Read_KB* This command reads the current KB filter setting. The command returns a single byte indicating if the KB filter is On or Off.	N/A	N/A	5	Data: 1 byte representing the state of the filter: 0: Off 1: On Ack: No
0x80000031	*Read_Model* This command reads the instrument model.	N/A	0-32	0-32	Data: ASCII string representing the Model. Size: Up to 32 bytes, including the termination byte. Ack: No
0x80000032	*Read_SN* This reads the serial number of the instrument	N/A	0-32	0-32	Data: ASCII string representing the serial number of the instrument. Size: Up to 32 bytes, including the termination byte Ack: No
0x80000033	*Read_FW_Rev* This command reads the firmware revision number.	N/A	0-32	0-32	Data: ASCII string representing the Firmware revision. Size: Up to 32 bytes, including the termination byte Ack: No

0x80000034	*Read_DOC* This command reads the date of last calibration. The UTC represents the number of seconds elapsed since Jan 1 1904.	N/A	N/A	8	Data: U64 number representing the UTC (Universal Time Code) of the date/time of last calibration. Ack: No
0x80000035	*Read_DOB* This command reads the date of birth of the instrument. The UTC represents the number of seconds elapsed since Jan 1 1904.	N/A	N/A	8	Data: U64 number representing the UTC (Universal Time Code) of the date/time of birth. Ack: No
0x80000036	*Read_User_ID* This command reads the User_ID field. That field can be written in persistent memory using the *Write_User_ID* command.	N/A	0-32	0-32	Data: ASCII string representing the User-ID, as defined by the user. Size: Up to 32 bytes, including the termination byte Ack: No
0x00000036	*Write_User_ID* This command writes the User_ID field in persistent memory.	N/A	0-32	1	Data: ASCII string representing the user-ID, as defined by the user. Size: Up to 32 bytes, including the termination byte Ack: Yes
0x80000050	*Read_Signal* This command reads the contents of the signal FIFO. The command requests N samples. The instrument returns the number N' of samples. Each sample is a	N/A	N	$(N' \times 3) + 4$	Data: 1 U32 representing the number N' of samples (triplets) that follows. $N' \times 3$ 32-bit IEEE-754 Floats, for X, Y

	triplet of float values, one value for each axis. The response will only contain the number of samples that are actually present in the signal FIFO. So $N' \leq N$. The response is: - An U32 value representing the number N' of samples actually sent back. - The specified number $N'$of triplets follows that value. Each member of a triplet is a *Sgl* float value. The scale of the signal is either m/s^2 for acceleration or m/s for velocity. The command can send back a maximum of 256 samples (triplets), so it is not useful to request more than 256 samples. Note. The FIFO can contain a maximum of 1024 samples (triplets) internally. That FIFO is typically filled continuously by the internal processing at the sampling frequency. So, when this command is sent the first few times, it will return stale data. It is necessary to request at least 1024 samples in order to get to newly measured data. The command must be sent repeatedly at a high enough rate relative to the sampling frequency to avoid missing data.				and Z axes. The 3 axes are interleaved in the response, starting with axis X. Ack: No

Table 3