Interfacing with VE.Bus products – MK2 Protocol

Table of contents

Interfacing with VE.Bus products – MK2 Protocol 1
1 Introduction 2
2 Supported devices 2
3 The MK2 interface 2
3.1 Communication parameters 2
3.2 Message format 3
3.3 Response times 4
3.4 Software compatibility 4
3.5 Jumpers in the MK2 5
3.6 MK2 Powerup sequence 6
4 LED status 6
5 Switch and input current settings 7
5.1 Receiving panel frames (0x40) 9
6 Special VE.Bus frames 10
6.1 Info frames (0x20) 10
6.2 MasterMultiLED frame (0x41) 12
6.3 Reset VE.Bus device frame 13
6.4 Battery Operational Limits 13
7 Communicating with a specific device 15
7.1 Device addresses 15
7.2 Device discovery 15
7.3 Winmon commands 15
7.3.1 CommandSendSoftwareVersion 18
7.3.2 CommandGetSetDeviceState 18
7.3.3 CommandReadRAMVar 19
7.3.4 CommandReadSetting 19
7.3.5 CommandWriteRAMVar 20
7.3.6 CommandWriteSetting 20
7.3.7 CommandWriteData 20
7.3.8 CommandGetSettingInfo 20
7.3.9 CommandGetRAMVarInfo 21
7.3.10 Units 22
7.3.11 CommandWriteViaID 22
7.3.12 CommandReadSnapShot 23
7.3.13 Setting and Variable IDs 23
Appendix 1 Simple example of common tasks 29
Appendix 2 Annotated example for typical UI 30
Appendix 3 Revision history 33

1 Introduction

This document describes the MK2 protocol, used to communicate with VE.Bus products. Note that implementing the MK2 protocol is a task which is not to be underestimated. It is a complicated protocol. There are other alternatives, with ModbusTCP being the most popular one. See our whitepaper 'Data communication with Victron Energy products' for more information:
http://www.victronenergy.com/support-and-downloads/whitepapers/

For communicating with older (VE 9-bit RS485) products, refer to "VE - Interfacing with the Phoenix Product Range".

2 Supported devices

The following Victron Energy products are equipped with VE.Bus connections:

- Victron Energy Phoenix Multi (Inverter/Charger)
- Victron Energy Phoenix Inverter
- Victron Energy Quattro

VE.Bus product version numbers are of the format AA BB CCC, where AA is the product type, BB is the model, and CCC is the version. The information in this document is intended for use with devices with software version xxyy111 or higher. (xx= 19,20 26 or 27)
Please make sure that your MK2 contains the latest firmware. This is automatically checked when connecting to a Multi/Quattro with latest version of VEConfigure3. (See also Software compatibility)

3 The MK2 interface

Communication with VE.Bus devices is achieved by using an MK2 interface (either MK2.2 or MK2 USB or MK3 USB). The MK2 provides a galvanically isolated serial connection.

Because the functionality of the Multis/Quattros is continuously expanding, it might be possible that the interface behavior of newer releases of Multis/Quattros is different in some aspects. An update of the MK2 firmware might be required to be able to communicate with newer Multis/Quattros. Always use the latest MK2 firmware available. Update it by connecting it to the latest version of VEConfigure3.

3.1 Communication parameters

Baud rate:	2400
Parity:	None
Data bits:	8
Stop bits:	1

Furthermore, when using a MK2.2 (this is the serial (non-USB) type), the DTR signal (pin 4 on the DB9 connector) must be driven high to provide power to the RS232 side of the MK2.
How to program the DTR is different 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.
Some 3rd party programs will set the DTR correct when RTS/CTS handshaking is enabled.
When the DTR level is not correct you will not be able to receive data from the MK2.2!

3.2 Message format

The basic frame format when communicating with the MK2 is:

<Length> 0xFF <Command> <Data$_0$> … <Data$_{n-1}$> <Checksum>

<Length> is the number of bytes in the frame, excluding the length and checksum bytes. If the MSB of <Length> is a 1, then this frame has LED status appended (see chapter 3.6 for more information). <Command> indicates the purpose of the frame. The number and content of the <Data> bytes depends on the value of <Command>. <Checksum> is computed such that the sum of all bytes in the frame (including the length and checksum) equals 0.

Please ignore the paragraph below. Update your Mk2 to the latest firmware version (see Software compatibility).
~~**Zero padding:**~~
~~If the last byte before <Checksum> is 0xFF, the frame will be interpreted differently by the MK2. To prevent this, if the last data byte of the frame is 0xFF, the frame must be padded with 0x00 (and <Length> incremented).~~
~~IMPORTANT: Do NOT add a 0x00 when the last byte <>0xFF~~

Notes:

- Please ignore this note, use latest MK2 firmware version
(see Software compatibility).
~~If the version of the MK2 is 1130128 or later then zero padding is not required.~~
~~Because of backwards compatibility reasons, an MK2 with version 1130128 or later, will allow zero padding to be used on commands which are supported by previous versions (all commands described in this document).~~
~~New commands, to be added to later versions in future, might not accept the zero padding!~~
~~Depending on the firmware version of the MK2, the response can also be padded with 0x00. Even if the last byte <> 0xFF.~~
~~(MK2 versions 1130128 and up do not use zero padding on their respones.)~~

- The MK2 protocol is little endian. All values that are larger than one byte are sent LSB first.

- Generally, only the <Command> and <Data> bytes will be discussed.

- To allow future enhancements, reponses can be extended in upcoming VE.Bus products resulting in extra data (bytes) to be added to responses. We will do our best to stay backwards compatible so the 'old' data (bytes) will still contain the info

as specified in this document.
To prevent problems with future VE.Bus products take care that your code allows for responses to become longer and that your code does not rely on values of non-documented bits.
Just ignore all non-documented superfluous data (bytes).

3.3 Response times

The response time of the MK2 will vary depending on the command. (For some commands the response value is known by the MK2 and so a response will be sent directly. Other commands require communication with one or more units on the VE.BUS so the response time for these commands will be increased)

We advise to use a timeout value of 500ms. (Note that response times will be much lower for the majority of the commands.)
This value is suitable for all commands with one exception: The response for `'F'5` (see paragraph 6) can take up to 750ms. So one should increase the timeout to a value greater than 750ms when waiting for that response.

3.4 Software compatibility

When the MK2 is powered up it will send a version frame. Version frames will also be sent approximately once per second if nothing else is being sent, and can also be requested with the following command:

Command: `'V'`
Reply: `'V'` `<Version number`$_0$`>` … `<Version number`$_3$`>` `<Mode>`

`<Version number>` is a 32-bit integer.
`<Mode>` Specifies the communications protocol in use. If `<Mode>` is 'W', the target device is communicating using the VE 9-bit RS485 protocol. Any other value for `<Mode>` means that the target is a VE.Bus device.
In that case `<Mode>` contains the address which is currently set in the MK2 (see paragraph 7.1) If no address is set then `<Mode>` is 'B'

The information in this document applies to MK2 versions of 1130125 or later, communicating in VE.Bus mode.
In new Multi/Quattro versions, communication mechanisms might change. The MK2 firmware will handle these changes for you. So:

It is important to always use the latest MK2/MK3 firmware !!

The firmware in an MK2/MK3 will be updated to the latest version automatically by connecting the MK2/MK3 to a PC running VE Configure and making a connection to a Multi.
(One should use an up to date version of VE Configure for this. VE Configure is available as a free download from www.victronenergy.com).

3.5 Jumpers in the MK2

Panel detect (VE.Bus pin 7, jumper J1 in the MK2).
This signal must be left floating or tied to GND. Connect this signal to ground to indicate to the device that a remote panel is connected. The device will not switch on until receives a command setting it to either on or charger-only.

Standby (VE.Bus pin 6, jumper J2 in the MK2)
This signal must be left floating or tied to GND. Connect this signal to ground to enable the power of the device when the "Panel detect" signal is low.

Device on/off switching, low power mode.
When the "Panel detect" signal is connected to GND the device will not switch on by just switching on the front switch. It will only then switch on when it receives a command to do so. If "Panel detect" is connected to GND and the device is off, the unit is in a low power mode[a]. The internal power supply is switched off in order to save energy. The device cannot send or receive commands in this state. To be able to switch it on one needs to pull the "Stand by" line to GND. This will enable the internal power supply and make it possible to receive and send commands. The "switch on" command can be send then and the unit will switch on. When the unit is switched on the "Stand by" line can be released. The unit itself will prevent the internal power to switch off. To switch the unit off, just send a command "switch off". It will switch off and the internal power supply will be switched off too. (Except when the "Stand by" line is still pulled to GND). One could also permanently tie the "Stand by" signal to GND if one wants to switch the device on and off via the interface. The advantage is that it is less complicated. The disadvantage is that in "off" mode the device will consume more power.

[a] To make use of the low power mode with a Multi Compact one must set dipswitch 2 to off.

3.6 MK2 Powerup sequence

On power-up, the MK2 will automatically detect if it is connected to a VE.Bus device or a VE 9-bit RS485 device. It will change the used baudrate and frame-types accordingly. It does that by listening for VE.Bus frames for a couple of 100 milliseconds. If it does not receive any, it defaults to VE 9-bit RS485 mode.

In case the MK2 is powered before the VE.Bus device, make sure to do one of the following:

A) Power cycle the MK2 after the VE.Bus device is powered.
B) Send the MK2 a reset command (`0x02 0xFF 'R' <checksum>`).

Note that this is only important in a non-standard system, where the MK2 is not powered from the VE.Bus device. When the MK2 is connected on to a Multi with a UTP cable, everything will be automatically, and this powerup sequence will not require extra attention or consideration.

4 LED status

The operating state of a VE.Bus system can be determined by requesting the LED status.

Command: `'L'`
Reply: `'L' <LED on> <LED blink> 0x00`[b]

Each bit in `<LED on>` represents the on/off status of an LED. Each bit in `<LED blink>` represents the blinking status of an LED; if the corresponding bit in `<LED on>` is 0 then the LED is blinking in anti phase with the others.

Bit number	LED
0	Mains
1	Absorption
2	Bulk
3	Float
4	Inverter
5	Overload
6	Low battery
7	Temperature

If the MK2 is unable to determine the LED status, `<LED on>` and `<LED blink>` will be reported as 0x1F.

[b] MK2 versions 1130128 and up do not send this 0x00

5 Switch and input current settings

The ‘S’ (state) command can be used to send the panel state (switch position and input current set point) to the VE.Bus system. Issuing this command will cause the panel state to be sent to VE.Bus (unless <Flags[4]> is set, see below). The MK2 can also be configured to automatically send the panel state every ½ second (<Flags[0]>).

There are 2 variants of this command, depending on (<Flags[7]>).

<u>**Variant 1**</u>: <Flags[7]>=0

Command: ‘S’ <Switch state> <Pot value> <Panel scale> 0x01 <Flags>
Reply: ‘S’

<Switch state>	Meaning
1	Charger only
2	Inverter only
3	On
4	Off

The <Panel scale> parameter is used to specify the maximum current the system should draw (in amps). The <Pot value> parameter can be any value between 0 and 255, and is used to allow the set point to be varied between 0 and <Panel scale> amps. The input current set point used by the system is determined by the following formula:

$$\text{Input current set point} = \frac{\texttt{<Pot value>} \times \texttt{<Panel scale>}}{256}$$

There are three exceptions, where this formula is not used:

<table>
<tr><th><Panel scale></th><th>Meaning</th></tr>
<tr><td>0</td><td>The input current set point will be set as low as possible.</td></tr>
<tr><td>1</td><td rowspan="2">The input current set point will be ignored by the system.</td></tr>
<tr><td>255</td></tr>
</table>

<u>**Variant 2**</u>: <Flags[7]>=1

Command: ‘S’ <Switch state> <Lo(Limit)> <Hi(Limit)> 0x01 <Flags> [<Reserved> <FlagsExt0> <EEPROM flags>]
Reply: ‘S’

The input current limit is in this case equal to Limit/10 Ampere.
<Switch state> is the same as in variant 1.
Making Limit>=0x8000 will result in the input current setpoint being ignored.

The parameters between brackets (“[ ]”) are optional.

Notes (for both variants):

1) If the input current set point exceeds the maximum rating of the equipment, it will be automatically limited to the maximum. The maximum rating of the equipment can be found by requesting a MasterMultiLED frame (see chapter 6.2 for more information).
2) When Powerassist is enabled there is a minimum value for the input current limit. This minimum value can be also be found by requesting a MasterMultiLED frame (see chapter 6.2 for more information). Specifying a value lower than this minimum limit (with exception of value 0 see next note) will effectively result in the minimum limit being used.
3) When the input current setpoint is set to zero one of 2 things happen:
    a. If Powerassist is enabled the Multi/Quattro will switch to Invert mode.
    b. If Powerassist is disabled, the charger will be disabled and the Multi/Quattro will switch to bypass mode.

`<Flags[n]>`	Meaning
0	Automatically send panel state to VE.Bus every ½ second.
1	Automatically append the LED status to all frames sent from the MK2 to the PC. If this flag is set, then all frames sent on the PC will have a LED status frame appended (see LED status above for details). When a frame has the LED status appended, bit 7 in the `<Length>` byte will be set.
2	Reserved (keep 0).
3	Reserved (keep 0).
4	Do not send panel state.
5	Reserved (keep 0).
6	Automatically forward received panel frames (see below).
7	0 = input current limit is send as potvalue and scale 1 = input current limit is send as Amps * 10

Optional:
`<Reserved>` (Keep 0)

`<FlagsExt0[n]>`	Meaning
0	0 = Winmon mode 2, long Winmon frames 1 = Winmon mode 1, short Winmon frames (convert long Winmon frames to old short type. (Default on)
1	Reserved (Keep 0).
2	1 = forward config responses from all devices
3	Ignore block mode requests

`<EEPROMFlags[n]>`	Meaning
0	Force VE.Bus mode (MK2 only). Skips autodetect VE.Bus/VE9bit mode
1	Block XMT. Listen only

5.1 Receiving panel frames (0x40)

If remote panel functionality is provided by another device in the system, the MK2 can be configured to forward the panel frames from that device, by setting `<Flags[6]>`.

Note: These frames are VE.Bus frames, *not* MK2 frames. The general format is the same, except that VE.Bus frames do not begin with 0xFF.

There are two types of panel frame, standard and extended. The type of frame sent depends on the panel model, and is indicated by bit 3 of `<Panel and switch information>`.

<Data byte>	Meaning	
	Standard	Extended
0	<Panel and switch information>	
1	<Pot value>	<Absolute current limit>
2	<Panel scale>	

`<Panel and switch information>`

Bit	7	6	5	4	3	2	1	0
Standard	<Panel ID>				<Frame format>	<Switch position>		
Extended						<Generator selected>	<Switch position>	

If `<Frame format>` is set, then the frame uses the extended format.

For standard format frames, `<Switch position>`, `<Pot value>` and `<Panel scale>` are interpreted as described above.

For extended format frames, add 1 to `<Switch position>` to get the standard format equivalent. `<Absolute current limit>` is the input current set point, specified in deci-amps.

6 Special VE.Bus frames

The ‘F’ command can be used to request information about the VE.Bus system, reset the VE.Bus system and communicate BOL frames.

Command: `'F' <Frame type>`
Reply: The format of the reply varies depending on the value of `<Frame type>`.

<Frame type>	Action	Reply format
0	Request DC info.	Info frame (DC)
1	Request AC L1 info.	Info frame (AC)
2	Request AC L2 info.	Info frame (AC)
3	Request AC L3 info.	Info frame (AC)
4	Request AC L4 info (rare).	Info frame (AC)
5	Request MasterMultiLED frame.	MasterMultiLED frame
6	Request snapshot of RAM variables of all connected Multis. Just append the RAM ID’s you are interested in (upto a maximum of 6 ID’s). Upon receive, each Multi in the system will make a copy of the current value of the requested RAM variables. These variables can be read out per device afterwards	You will not get a reply on this command. When you perform a ‘readout’ of the snapshots on the separate Multis you will see whether or not this Multi has taken a snapshot.
7	Reserved	
8	Reset VE.Bus device	None
9	Send Battery Operational Limits	BOL response Frame

Note: The responses to the ‘F’ command are VE.Bus frames, *not* MK2 frames. The general format is the same, except that VE.Bus frames do not begin with 0xFF.

6.1 Info frames (0x20)

<Data byte>	Meaning	
	DC frame	AC frame
0	Reserved	<BF factor>
1		<Inverter factor>
2	Reserved	Reserved
3	Reserved	<State>
4	<Phase info>	
5	<DC voltage>	<Mains voltage>
6		
7	<DC current used by inverting devices>	<Mains current>
8		
9		<Inverter voltage>
10	<DC current provided by	
11		<Inverter current>

12	charging devices>	
13	<Inverter period>	<Mains period>

The voltage, current and period fields use the same offsets and scales as the equivalent values requested with 'W' commands (see chapter 7.3 for more information). Additionally, <Mains current> must be multiplied by <BF factor> to get the total mains current for that phase, and <Inverter current> must be multiplied by <Inverter factor>[c] to get the total inverter current for that phase.

Note: The DC current fields are unsigned 24-bit values.

<Phase info> indicates which phase the received frame describes, and in the case of L1, how many phases are present in the system.

<Phase info>	Meaning
0x05	This frame describes L4.
0x06	This frame describes L3.
0x07	This frame describes L2.
0x08	This frame describes L1; there is 1 phase in this system.
0x09	This frame describes L1; there are 2 phases in this system.
0x0A	This frame describes L1; there are 3 phases in this system.
0x0B	This frame describes L1; there are 4 phases in this system.
0x0C	This is a DC info frame.

<state> indicates the Multi main state. This is the same state as returned by the separate Winmon command CommandGetSetDeviceState 7.3.2 .The <state> field is available in Multi firmware versions >= 207

<State>	Meaning
0x0	Down
0x1	Startup
0x2	Off
0x3	Slave
0x4	InvertFull
0x5	InvertHalf
0x6	InvertAES
0x7	PowerAssist
0x8	Bypass
0x9	StateCharge

[c] Inverters (not Multis) with software versions xxyy120 up to and including xxyy125, (xx being 19,20,26 or 27) will incorrectly report <Inverter factor> as 0. Updating the device to revision 126 or higher will fix this.

6.2 MasterMultiLED frame (0x41)

<Data byte>	Meaning
0	Reserved
1	
2	
3	
4	<AC input configuration>
5	<Minimum input current limit>
6	
7	<Maximum input current limit>
8	
9	<Actual input current limit>
10	
11	<Switch register>

The <* input current limit> fields are in deci-amps.

<AC input configuration> can be further broken down as follows:

<AC input configuration>	Meaning	From version
0	<Last active input>	
1		
2	<Input current overridden by panel>	
3	<DMC dedicated. If high ignore bytes 4..10>	
4	<Nr of ACInputs in the system>	415
5		
6		
7	<Remote panel detected>	415

<Last active input> indicates which of the AC inputs was the last to be used. Note that this does not necessarily mean that there is currently anything connected to this input. The first input is input 0.

If <Input current limit overridden by panel> is set, then a remote panel can override the internal input current limit for this input. If this bit is not set, then the internal setting will be used even if a remote panel is connected.

<Switch register> can be further broken down as follows:

<Switch register>	
0	<DirectRemoteSwitchCharge> panel
1	<DirectRemoteSwitchInvert> panel
2	<FrontSwitchUp>
3	<FrontSwitchDown>
4	<SwitchCharge> current state
5	<SwitchInvert> current state

6	<OnboardRemoteInvertSwitch>
7	<RemoteGeneratorSelected>

6.3 Reset VE.Bus device frame

VE.Bus devices can be reset individually or all at once.

The special VE.Bus frame has the following format:

Command: 'F' 8 <ID$_{2}$> <ID$_{3}$> <ID$_{0}$> <ID$_{1}$>
Reply: None

<ID> contains the long address of the VE.Bus device. If 0, all devices will be reset.

6.4 Battery Operational Limits

Requires Multi firmware version $\geq$ 415.
Requires MK3 firmware version $\geq$ 203.
MK2 not supported.

Battery Operational Limits (BOL) should only be used to implement BMS functionality. The following applies:

1. When sending BOL frames the Multi's BOL engine starts.
2. A communication timeout of 5 minutes causes charging/inverting to stop
3. Once enabled the BOL engine keeps running until the next Multi reset
4. BOL is not for lead batteries (only bulk and absorption is supported)

The special VE.Bus frame has the following format:

Command: 'F' 9
<Flags>
<UBatCharge$_{0}$> <UBatCharge$_{1}$>
<IBatCharge$_{0}$> <IBatCharge$_{1}$> <IBatCharge$_{2}$>
<UBatDischarge$_{0}$> <UBatDischarge$_{1}$>
<IBatDischarge$_{0}$> <IBatDischarge$_{1}$> <IBatDischarge$_{2}$>

Response: 0x3C
<Flags>
<UBatCharge$_{0}$> <UBatCharge$_{1}$>
<IBatCharge$_{0}$> <IBatCharge$_{1}$> <IBatCharge$_{2}$>
<UBatDischarge$_{0}$> <UBatDischarge$_{1}$>
<IBatDischarge$_{0}$> <IBatDischarge$_{1}$> <IBatDischarge$_{2}$>

The <Flags> bits define the contents and thereby the length of the frame and are specified as follows:

<Flags[n]>	Meaning
0	UBat charge present
1	IBat charge present
2	UBat discharge present

3	IBat discharge present
7	Only used in response. Info missing (Timeout occured)

The `<UBatCharge>,<UBatDischarge>` fields are unsigned 16-bit values (unit cV).
The `<IBatCharge>,<IBatDischarge>` fields are unsigned 24-bit values (unit dA).

Note: If `<IBatDischarge>` equals 0 discharging will be stopped. All other values allow discharge.

7 Communicating with a specific device

All commands described so far are for interacting with the entire VE.Bus system. Some features however, are specific to each device in the system. The 'W' command (described below) is used for communicating with a specific device.

7.1 Device addresses

In a VE.Bus system, there may be more than one target device. In order to communicate directly with a specific target, you must tell the MK2 the address of the device with which you wish to communicate. This is done using the 'A' (address) command which has the following format:

Command: `'A' <Action> <Address>`
Reply: `'A' <Action> <Address> 0x00`[d]

Bit 0 of the `<Action>` parameter determines whether the `<Address>` parameter should be read or written. If this bit is a 1 then the address used by the MK2 will be set to the value of the `<Address>` field, otherwise the `<Address>` field is ignored. The reply will return the address currently in use. The `<Action>` parameter in the reply will be the same value specified in the command frame.

When the MK2 starts up, the address will be set to the default value of 0xFF. This is not a valid address, so must be set before it is possible to send 'W' commands (see chapter 7.3), a valid address must be set. Valid address values are between 0x00 and 0x1F. Attempting to set any other value will cause the MK2 to revert to the default value of 0xFF.

When sending 'W' commands that should generate a response, you must wait for the response before sending another 'W' or 'A' command, otherwise the response will not be received.

7.2 Device discovery

In order to set the correct address, you must first determine the address(es) of the target device(s). To do this, set an address then send any 'W' command that will cause the target to respond. If a response is received, then a valid address is selected, otherwise there is no device at that address. Begin with address 0, and test each address in turn. If no reply is received for 3 consecutive addresses, then there are no more devices and it is not necessary to continue.

7.3 Winmon commands

Currently two Winmon cmd modes are supported.

Modes:

[d] MK2 versions 1130128 and up do not send this 0x00

Mode	Description	Note
1 (short/legacy)	command with 2 bytes payload	***MK2 default***
2 (long)	command with upto 14 bytes payload	Can be selected with ‘S’command (Requires Multi firmware versions 2xx, 3xx, 4xx)

By default, for backwardscompatibility the MK2 translates incoming long mode 2 frames into short mode 1 frames.

Winmon Commands:

<WXYZ>	
‘W’	Original Winmon command with 2 bytes payload.
‘X|Y|Z’	Successive commands rotate 'x[X]','y[Y]','z[Z]' in order to be able to distinguish which reply belongs to what command.
‘x|y|z’	In addition to above this includes the destination address at the end of the frame. Winmon replies are always uppercase.

The ‘W’ command

Command: ‘W’ <W frame>
Response: ‘W’ <W frame$_0$> [<W frame$_1$> [<W frame$_2$>]]

When sending ‘W’ commands, the response from the target device may consist of multiple <W frame>s. In this case, up to 3 <W frame>s will be packed into one MK2 frame.

The format of a <W frame> is as follows:

<W command> <Info$_0$> <Info$_1$>

Each <W frame> will always be 3 bytes, so the number of frames included in a single response to a ‘W’ command can be determined from the value of <Length>.

The ‘X|Y|Z’ command

‘X’ or ‘Y’ or ‘Z’ replace the ‘W’ command in order. This enables us to distinguish which reply belongs to what command.

Command: ‘X’ <W frame>
‘Y’ <W frame>
‘Z’ <W frame>
Response (mode 1): ‘X|Y|Z’ <W frame$_0$> [<W frame$_1$> [<W frame$_2$>]]

The format of a <W frame> is as follows:

<W command> <Info$_0$> <Info$_1$>

Response (mode 2): 'X|Y|Z' <W command> <Info$_0$> [<Info$_1$> [<Info$_2$> [<info$_x$>]]]

The 'x|y|z' command

This version includes the destination address

Command: 'x' <W frame> <addr>
'y' <W frame> <addr>
'z' <W frame> <addr>

Response (mode 1): 'X|Y|Z' <W frame$_0$> [<W frame$_1$> [<W frame$_2$>]]

The format of a <W frame> is as follows:

<W command> <Info$_0$> <Info$_1$>

Response (mode 2): 'X|Y|Z' <W command> <Info$_0$> [<Info$_1$> [<Info$_2$> [<info$_x$>]]]

From this point forward, only the contents of the <W frame> part of the command will be discussed.

Possible Winmon commands are:

W command	Name	Note
0x05	CommandSendSoftwareVersionPart0	
0x06	CommandSendSoftwareVersionPart1	
0x0E	CommandGetSetDeviceState	
0x30	CommandReadRAMVar	
0x31	CommandReadSetting	
0x32	CommandWriteRAMVar	
0x33	CommandWriteSetting	
0x34	CommandWriteData	
0x35	CommandGetSettingInfo	
0x36	CommandGetRAMVarInfo	
0x37	CommandWriteViaID	Winmon mode 2 only
0x38	CommandReadSnapShot	

If an unsupported command is sent, the device will reply with an "Unknown command" response:

Response: 0x80 <Reason> <XX>

<Reason> indicates why the command was unrecognised. If bit 0 is set, <Info$_0$> was not recognised. If bit 1 is set, <Info$_1$> was not recognised. If both bits are clear, <W command> was not recognised.

7.3.1 CommandSendSoftwareVersion

The software version of the target device is a 4-byte integer which can be requested with

Mode 1 (short Winmon frames):
CommandSendSoftwareVersionPart0 (low bytes) and
CommandSendSoftwareVersionPart1 (high bytes).

Command(s): 0x05/0x06 XX XX
Response : 0x82 <Lo$_{Part0}$> <Hi$_{Part0}$>
0x83 <Lo$_{Part1}$> <Hi$_{Part1}$>

Mode 2 (long Winmon frames):
CommandSendSoftwareVersionPart0

Command: 0x05
Response : 0x82 <Lo$_{Part0}$> <Hi$_{Part0}$> <Lo$_{Part1}$> <Hi$_{Part1}$> XX XX

7.3.2 CommandGetSetDeviceState

This command is used to read the state of the device or to force the unit to go into a specific state.

Command: 0x0E <State> XX
Response: 0x94 <State> <Sub-state>

Command

<State>	Action
0	No state change, just inquire.
1	Force to Equalise. 1 hour 1, 2 or 4 V above absorption (12/24/48V). Charge current is limited to ¼ of normal value. Will be followed by a normal 24-hour float state.
2	Force to Absorption, for maximum absorption time. Will be followed by a normal 24-hour float state.
3	Force to Float, for 24 hours.

Response

<State>	State description	Sub-state	Sub-state description
0	Down	0	
1	Startup	0	
2	Off	0	
3	Device in slave mode	0	
4	Invert Full	0	
5	Invert Half	0	
6	Invert AES	0	
7	Power Assist	0	
8	Bypass	0	
9	Charge	0	Charge Initializing
		1	Charge Bulk
		2	Charge Absorption

		3	Charge Float
		4	Charge Storage
		5	Charge Repeated Absorption
		6	Charge Forced Absorption
		7	Charge Equalise
		8	Charge Bulk stopped

Note: Switching the state might take some time so it is possible that the returned state does not correspond directly with the requested state. However if state change is possible it will take place within 1 second. So depending on the application a verify might be needed.

If the requested state is not supported then an "Unknown command" response is sent.

Note: This command was introduced with firmware versions xxyy122 (xx= 19,20,26 or 27)
Setting "force equalize", "force absorption and "force float" works form xxyy400

7.3.3 CommandReadRAMVar

This command can be used to read RAM variables. A list of RAM IDs can be found in chapter 7.3.11. Not all devices support all variables, refer to CommandGetRAMVarInfo for information on how to determine which variables are supported, and how to interpret them.

Command: 0x30 <RAM ID_0> [<RAM $ID_{1..5}$>]
Response: 0x85/0x90 <Lo(Value)> <Hi(Value)>

0x85 = RamReadOK.
0x90 = Variable not supported (in which case <Value> is not valid).

By adding more IDs to the command, up to six RAM variables can be requested in a single command.

7.3.4 CommandReadSetting

This command can be used to read device settings. A list of setting IDs can be found in chapter 7.3.11. Not all devices support all settings, refer to CommandGetSettingInfo for information on how to determine which settings are supported, and how to interpret them.

Command: 0x31 <Lo(Setting ID)> <Hi(Setting ID)>
Response: 0x86/91 <Lo(Value)> <Hi(Value)>

<Value> is an unsigned 16-bit quantity.

0x86 = SettingReadOK.
0x91 = Setting not supported (in which case <Value> is not valid).

7.3.5 CommandWriteRAMVar

This command can be used to write RAM variables. A list of RAM IDs can be found in chapter 7.3.11. Not all devices support all variables, refer to CommandGetRAMVarInfo for information on how to determine which variables are supported, and how to represent them.

Command: `0x32 <Lo(RAM ID)> <Hi(RAM ID)>`
Response: None

This command must be followed by CommandWriteData.

7.3.6 CommandWriteSetting

This command can be used to write settings. A list of setting IDs can be found in chapter 7.3.11. Not all devices support all settings, refer to CommandGetSettingInfo for information on how to determine which settings are supported, and how to represent them.

Command: `0x33 <Lo(Setting ID)> <Hi(Setting ID)>`
Response: None

This command must be followed by CommandWriteData.

7.3.7 CommandWriteData

This command must be used in conjunction with either CommandWriteRAMVar or CommandWriteSetting. This command sends the data to be written. The destination of the data depends on the previous frame.

Command: `0x34 <Lo(Data)> <Hi(Data)>`
Reply: `0x87/0x88 XX XX`

0x87 = successful RAM write.
0x88 = successful setting write.

7.3.8 CommandGetSettingInfo

This command can be used to get information about which settings are supported, and how to interpret them. A list of setting IDs can be found in chapter 7.3.11.

Mode 1 (short Winmon frames):

Command: `0x35 <Lo(Setting ID)> <Hi(Setting ID)>`
Responses:

```
0x89 <Lo(Sc)> <Hi(Sc)>
0x8A <Lo(Offset)> <Hi(Offset)>
0x8B <Lo(Default)> <Hi(Default)>
0x8C <Lo(Minimum)> <Hi(Minimum)>
0x8D <Lo(Maximum)> <Hi(Maximum)>
```

If Sc = 0, this setting is not supported, and the remaining responses are not transmitted.

Mode 2 (long Winmon frames):

Command: 0x35 <Lo(Setting ID)> <Hi(Setting ID)>
Responses: 0x89 <Lo(Sc)> <Hi(Sc)> <Lo(Offset)> <Hi(Offset)>
<Lo(Default)> <Hi(Default)> <Lo(Minimum)>
<Hi(Minimum)> <Lo(Maximum)> <Hi(Maximum)>
<Access level>

If the response is 0x86 without data. This setting is not supported

<Sc> is a signed 16-bit value.

The scaling factor (Scale) can be determined from Sc as follows:

If Sc > 0 then Scale := Sc
else Scale := 1 / (-Sc)

Note: The interpretation of Sc is different for RAM variables.

<Offset> is a signed 16-bit value.

The scale and offset are used to format the setting value. Assume x is the (16-bit) value of a setting (requested with CommandReadSetting). The following formula is used to determine the display value:

DisplayValue := Scale * (x + Offset)

<Default>, <Minimum>, <Maximum> are unsigned 16-bit values, represented in the same format as the values returned by CommandReadSetting. To be meaningful for the end-user these values must be formatted with Scale and Offset as above.

7.3.9 CommandGetRAMVarInfo

This command can be used to get information about which RAM variables are supported, and how to interpret them. A list of RAM IDs can be found in chapter 7.3.11.

Mode 1 (short Winmon frames):
Command: 0x36 <Lo(RAM ID)> <Hi(ID)>
Responses: 0x8E <Lo(Sc)> <Hi(Sc)>
0x8F <Lo(Offset)> <Hi(Offset)>

If Sc = 0, this RAM variable is not supported, and the remaining response is not transmitted.

Mode 2 (long Winmon frames):
Command: 0x36 <Lo(RAM ID)> <Hi(ID)>
Responses: 0x8E <Lo(Sc)> <Hi(Sc)> <Lo(Offset)> <Hi(Offset)>

<Sc> is a signed 16-bit value.

If Sc < 0 then the 16-bit value returned by CommandReadRAMVar is signed.
If Sc > 0 then the 16-bit value returned by CommandReadRAMVar is unsigned.

The scaling factor (Scale) can be determined from Sc as follows:

Scale := Abs(Sc)
If Scale ≥ 0x4000
    Scale := 1 / (0x8000 – Scale)

Note: The interpretation of Sc is different for settings.

<Offset> is a signed 16-bit value.

The scale and offset are used to format the RAM variable. Assume x is the (16-bit) value of a RAM variable (requested with CommandReadRAMVar). The following formula is used to determine the display value:

DisplayValue := Scale * (x + Offset)

Note: x could either be a signed or unsigned value, depending on <Sc>.

Special case:

When <Offset> is 0x8000, the RAM variable is a bit. <Sc> is then set to the bit number + 1 in that case.

7.3.10 Units

Unless otherwise noted, values use the following units:

Type	Unit
Voltage	Volt
Current	Ampere
Time	Minute

7.3.11 CommandWriteViaID

This command can be used to write RAM variables or settings.

Command: 0x37 <flags> <ID> <Lo(data)> <Hi(data)>
Response: 0x80/0x87/0x88/0x9B

0x80 = Command not supported
0x87 = Write ramvar OK
0x88 = Write setting OK
0x9B = Access level required

<flags>

Bit	7	6	5	4	3	2	1	0
Meaning							(0)update ram and EEPROM (1)ram value only	(0)RAMvar (1)setting
Remark							Firmware versions >= 412	

7.3.12 CommandReadSnapShot

This command can be used to read the current snapshot.

Command: 0x38
Response: 0x99, [<Lo(V0)> <Hi(V0)> [<Lo(V1)> <Hi(V1)> [Lo(V2)> <Hi(V2)> [Lo(V3)> <Hi(V3)> [Lo(V4)> <Hi(V4)> [Lo(V5)> <Hi(V5)>]]]]]]

The device will respond with the values of the last snapshot (if any).
A snapshot should be requested using the 'F' command and can only be read once.

Notes:
1) Taking a snapshot is especially useful when you would like to have for instance the power of a system as accurate as possible. You should take a snapshot of the non-filtered values and calculate the total sum of all devices. It might be useful to apply some filtering on this summed value.
2) The Multis/Quattros have only 1 snapshot memory.
When a new snapshot is requested before the old has been read the old will be lost!
Please note that the CCGX might send snapshot commands so you cannot safely use this mechanism in a system with a CCGX!!

7.3.13 Setting and Variable IDs

7.3.13.1 RAM variables

ID	Function
0	UMainsRMS
1	IMainsRMS
2	UInverterRMS
3	IInverterRMS
4	UBat
5	IBat
6	UBatRMS (= RMS value of ripple voltage)
7	Inverter Period Time (time-base 0.1s)
8	Mains Period Time (time-base 0.1s)
9	Signed AC Load Current
10	Virtual switch position
11	Ignore AC input state
12	Multi functional relay state
13	Charge state (battery monitor function)
14	Inverter Power (filtered) Power being transformed from AC to DC or vice versa This is a 16 bit signed integer. Positive power means energy converted from AC to DC (i.e. charging) negative power means conversion from DC to AC (i.e. discharging, inverting)

15	Inverter Power (filtered) Power being transformed from AC to DC or vice versa This is a 16 bit signed integer. Positive power means energy converted from AC to DC (i.e. charging) negative power means conversion from DC to AC (i.e. discharging, inverting)
16	Output power (filtered) Power going into or out of the Multi/Quatrro on the AC output port to the loads. This is a 16 bit signed integer. Positive power means energy flowing out of the Multi/Quattro towards the load, negative means power going from the "loads" into the Multi/Quattro.
17-19	17-19 correspond with 14-16 but these are the 'unfiltered' values. These IDs return the power measured during the last cycle. (In most cases you will get better results using the filtered values.)

7.3.13.2 Settings

ID	Function
0	Flags$_0$ (see below)
1	Flags$_1$ (see below)
2	UBatAbsorption
3	UBatFloat
4	IBatBulk
5	UInvSetpoint
6	IMainsLimit (AC1)
7	Repeated Absorption Time
8	Repeated Absorption Interval
9	(Maximum) Absorption duration
10	Charge characteristic
11	UBatLowLimit for Inverter
12	UBatLow hysteresis for Inverter
13	Number of slaves connected
14	Special three phase setting 0 = 3 phase 1 = Split phase 180 2= 2 leg 3 phase 120
15-43	Used for Virtual Switch (see Virtual switch settings below)
44	Lowest acceptable UMains
45	Hysteresis for parameter 44
46	Highest acceptable UMains
47	Hysteresis for parameter 46
48	Assist current boost factor
49	IMainsLimit (AC2)
50	Low current limit for switching to AES[e]
51	Hysteresis on AES current limit[e] (leave AES when current > Settings 50+51)
52-59	Used for Virtual Switch (see Virtual switch settings below)
60	Flags$_2$ (see below)
61	Flags$_3$ (see below)
62	Used for Virtual Switch (see Virtual switch settings below)
63	UBat low pre-alarm offset. Must be added to (UBatLowLimit + UBatLowHysteresis) to determine the pre-alarm level. This offset can be positive or negative. Since settings are only positive 0x8000 is added to the value.
64	Battery capacity for battery monitor function.
65	For battery monitor function. Specifies the charge percentage to which battery status is set when the charge state changes from Bulk to Absorption.

7.3.13.3 Flags

To determine which flags are supported by the device, the `<Maximum>` value returned by CommandGetSettingInfo is used. In the case of the flag settings, `<Maximum>` is a bit mask, where a bit will be set for each supported setting.

[e] The setting will become active after a reset of the Multi (off/on with the front switch or with a remote panel which releases the Stand By signal).

Flags[0] is bit 0 in $Flags_0$, Flags[31] is bit 15 in $Flags_1$

Flags[n]	Function
0	MultiPhaseSystem
1	MultiPhaseLeader
2	60Hz
3	Disable Wave Check (fast input voltage detection). **IMPORTANT**: Keep flags[7] consistent.
4	DoNotStopAfter10HrBulk
5	AssistEnabled
6	DisableCharge
7	**IMPORTANT**: Must have inverted value of flags[3].
8	DisableAES
9	Not promoted option
10	Not promoted option
11	EnableReducedFloat
12	Not promoted option
13	Disable ground relay
14	Weak AC input
15	Remote overrules AC2
16-26	Virtual switch flags (see below)
27	Accept wide input frequency
28	Dynamic current limiter
29	Use tubular plate traction battery curve
30	Remote overrules AC1
31	Use Low Power Shutdown in AES instead of modified sinewave.
32-34	Virtual switch flags (see below)
35-63	Unused

Warning: do not change unused flags. The result may be unpredictable.

Warning: do not change ‘Not promoted options’. Changing these can damage the device.

Warning: When changing flags, do not set bits for unsupported settings, as this can cause the values of other flags to be changed.

7.3.13.4 Virtual switch settings

ID	Name	Detail
15	vsUsage	0: Not used; 1: Use VS to control Relay; 2: Use VS to ignore AC input
16	vsonIInvHigh	Level
17	vsonUBatHigh	Level
18	vsonUBatLow	Level
19	vstonIInvHigh	Time
20	vstonUBatHigh	Time
21	vstonUBatLow	Time
22	vstonNotCharging	Time
23	vstonFanOn	Time
24	vstonTemperatureAlarm	Time
25	vstonLowBatteryAlarm	Time
26	vstonOverloadAlarm	Time
27	vstonUBatRippleAlarm	Time
28	vsoffIInvLow	Level
29	vsoffUBatHigh	Level
30	vsoffUBatLow	Level
31	vstoffIInvLow	Time
32	vstoffUBatHigh	Time
33	vstoffUBatLow	Time
34	vstoffCharging	Time
35	vstoffFanOff	Time
36	vstoffChargeBulkFinished	Time
37	vstoffNoVSOnCondition	Time
38	vstoffNoACInput	Time
39	vstoffTemperatureAlarm	Time
40	vstoffLowBatteryAlarm	Time
41	vstoffOverloadAlarm	Time
42	vstoffUBatRippleAlarm	Time
43	vsMinimumOnTime	Time; will not influence Off conditions with time set to 0
52	vs2onILoadHigh	Level
53	vs2onILoadHigh	Time
54	vs2onUBatLow	Level
55	vs2onUBatLow	Time
56	vs2offILoadLow	Level
57	vs2offILoadLow	Time
58	vs2offUBatHigh	Level If high byte is 0 then if: low byte=0 Condition is "When Bulk finished" low byte=1 Condition is "When Abs. finished"
59	vs2offUBatHigh	Time
62	vsInverterPeriodTime	

7.3.13.5 Virtual switch flags

Flags[n]	Name	Detail
16	vsonBulkProtection	

17	vsonTemperaturePreAlarm	
18	vsonLowBatteryPreAlarm	
19	vsonOverloadPreAlarm	
20	vsonUBatRipplePreAlarm	
21	vsoffTemperaturePreAlarm	
22	vsoffLowBatteryPreAlarm	
23	vsoffOverloadPreAlarm	
24	vsoffUBatRipplePreAlarm	
25	vsonWhenGeneralFailure	
26	vsInvert	When on: When a VS On condition is met, the relay is deactivated, or AC input is NOT ignored.
32	vs2offWhenAC1Available	
33	vs2Invert	
34	vsSetInverterPeriodTime	

The names are closely related to the function. For example, vstonUBatHigh means the time UBat must be above vsonUBatHigh before VS is considered on.

Note: On conditions have priority over off conditions.

Appendix 1 Simple example of common tasks

Here are some sample messages for performing common tasks. Messages to the MK2 are shown in `green`, messages from the MK2 are shown in `blue`. Parentheses are used to group data bytes belonging to the same message field. Where the value of a byte depends on the connected device/running state it is shown as `XX`. Where the whole message is known, the checksum has been calculated; otherwise it is shown as <Checksum>. 'Quoted' bytes are ASCII representations all other values are hexadecimal representations.

Instruct the MK2 to communicate with the VE.Bus device at address 0

04 FF 'A' 01 00 BB
05 FF 'A' 01 00 00[f] BA

Read the software version of the target

05 FF 'W' 05 (00 00) A0
06 FF 'W' 82 (XX XX) 00[f] <Checksum>

05 FF 'W' 06 (00 00) 9F
06 FF 'W' 83 (XX XX) 00[f] <Checksum>

Request the scale and offset information of the DC voltage

05 FF 'W' 36 (04 00) 6B
09 FF 'W' 8E (XX XX) 8F (XX XX) <Checksum>

Request the State of Charge value

05 FF 'W' 30 0D 00 68
05 FF 'W' 85 (XX XX)[DD DD][g] <Checksum>

Request the DC info frame

03 FF 'F' 00 B8
0F 20 (XX XX XX XX) 0C (XX XX) (XX XX XX) (XX XX XX) XX <Checksum>

Request the LED status

02 FF 'L' B3
05 FF 'L' XX XX 00[f] <Checksum>

Instruct the MK2 to act as a remote panel - switch on, input current limit 12A (of 16A max.) using variant 1 (sending potvalue and scale)

07 FF 'S' 03 C0 10 01 01 D2
02 FF 'S' AC

Send single remote panel command - switch on, input current limit 31.5A using variant 2 (sending absolute current limit)

07 FF 'S' 03 3B 01 01 80 E7
02 FF 'S' AC

Instruct the MK2 to append LED status to all frames

07 FF 'S' XX XX XX 01 02 <Checksum>
84 FF 'S' XX XX <Checksum>

---

[f] MK2 versions 1130128 and up do not send this 0x00
[g] Newer VE.Bus products add the [DD DD] this can be ignored see also note in §3.2

Appendix 2 Annotated example for typical UI

Note: checksums are not shown in the dump!

First make sure that the MK2 is connected

```
00.276 INF.mk2:      Getting version
00.338 INF.:   > 02 FF V
00.401 INF.:   < 07 FF V 93 3E 11 00 00
00.401 INF.task:     version frame received
```

In order to obtain the scaling a valid device must be selected, in a properly configured system, address 0 will always be used, set that:

```
00.401 INF.mk2:      setting address: 00
00.448 INF.:   > 04 FF A 01 00
00.510 INF.:   < 04 FF A 01 00
```

Get the ram scaling needed to intepreted the values from the device.

```
00.557 INF.:   > 05 FF W 36 00 00
00.650 INF.:   < 08 FF W 8E 9C 7F 8F 00 00
00.650 INF.winmon:  ram info scale=32668 offset=0
00.650 INF.init:     scale for 0 = 0.010000
00.697 INF.:   > 05 FF W 36 01 00
00.806 INF.:   < 08 FF W 8E 9C 7F 8F 00 00
00.806 INF.winmon:  ram info scale=32668 offset=0
00.806 INF.init:     scale for 1 = 0.010000
00.884 INF.:   > 05 FF W 36 02 00
00.978 INF.:   < 08 FF W 8E 9C 7F 8F 00 00
00.978 INF.winmon:  ram info scale=32668 offset=0
00.978 INF.init:     scale for 2 = 0.010000
01.056 INF.:   > 05 FF W 36 03 00
01.165 INF.:   < 08 FF W 8E 9C 7F 8F 00 00
01.165 INF.winmon:  ram info scale=32668 offset=0
01.165 INF.init:     scale for 3 = 0.010000
01.212 INF.:   > 05 FF W 36 04 00
01.306 INF.:   < 08 FF W 8E 9C 7F 8F 00 00
01.306 INF.winmon:  ram info scale=32668 offset=0
01.306 INF.init:     scale for 4 = 0.010000
01.352 INF.:   > 05 FF W 36 05 00
01.446 INF.:   < 08 FF W 8E 64 80 8F 00 00
01.446 INF.winmon:  ram info scale=-32668 offset=0
01.446 INF.init:     scale for 5 = 0.010000
01.508 INF.:   > 05 FF W 36 06 00
01.602 INF.:   < 08 FF W 8E 9C 7F 8F 00 00
01.602 INF.winmon:  ram info scale=32668 offset=0
01.602 INF.init:     scale for 6 = 0.010000
01.664 INF.:   > 05 FF W 36 07 00
01.758 INF.:   < 08 FF W 8E 57 78 8F 00 01
01.758 INF.winmon:  ram info scale=30807 offset=256
01.758 INF.init:     scale for 7 = 0.000510
```

```
01.820 INF.:   > 05 FF W 36 08 00
01.914 INF.:   < 08 FF W 8E 2F 7C 8F 00 00
01.914 INF.winmon:  ram info scale=31791 offset=0
01.914 INF.init:    scale for 8 = 0.001024
01.961 INF.:   > 05 FF W 36 09 00
02.054 INF.:   < 08 FF W 8E 64 80 8F 00 00
02.054 INF.winmon:  ram info scale=-32668 offset=0
02.054 INF.init:    scale for 9 = 0.010000
02.117 INF.:   > 05 FF W 36 0A 00
02.195 INF.:   < 08 FF W 8E 04 00 8F 00 80
02.195 INF.winmon:  ram info scale=4 offset=-32768
02.195 INF.init:    10 is a bit field, bit 3
02.382 INF.:   > 05 FF W 36 0B 00
02.460 INF.:   < 08 FF W 8E 01 00 8F 00 80
02.460 INF.winmon:  ram info scale=1 offset=-32768
02.460 INF.init:    11 is a bit field, bit 0
02.491 INF.:   > 05 FF W 36 0C 00
02.600 INF.:   < 08 FF W 8E 06 00 8F 00 80
02.600 INF.winmon:  ram info scale=6 offset=-32768
02.600 INF.init:    12 is a bit field, bit 5
02.647 INF.:   > 05 FF W 36 0D 00
02.741 INF.:   < 08 FF W 8E 38 7F 8F 00 00
02.741 INF.winmon:  ram info scale=32568 offset=0
02.741 INF.init:    scale for 13 = 0.005000
02.741 INF.vebus_startup:   settings scale obtained
```

==== Current limit + active input ====
Get the current limit info, these have fixed units of 0.1A.

```
02.788 INF.:   > 03 FF F 05
03.193 INF.:   < 0C 41 10 09 00 00 00 3E 00 E8 03 F4 01
03.193 INF.ac-in-config:    active input: 0
03.193 INF.ac-in-config:    raw current limit 500 [62 - 1000]
03.193 INF.mk2-values:      ac in limit: 50.0A [6.2A .. 100.0A]
```

==== DC values + inverter frequency ====
AC values need to be scaled with the scales obtained above:

```
out->V = mk2UnpackRamFloat(raw->V, &dev->ramInfo[WM_VAR_UBAT]);
out->I = mk2UnpackRamFloat(raw->chargerI, &dev->ramInfo[WM_VAR_IBAT]);
out->I -= mk2UnpackRamFloat(raw->inverterI, &dev->ramInfo[WM_VAR_IBAT]);

/* convert period to frequency in Hz */
out->frequency = mk2UnpackRamFloat(raw->period, &dev-
>ramInfo[WM_VAR_INVERTER_PERIOD_TIME]);
if (out->frequency)
        out->frequency = 10 / out->frequency;
```

```
03.349 INF.:   > 03 FF F 00
03.474 INF.:   < 0F 20 7F 9A 81 79 0C 51 0A 00 00 00 00 00 00 88
```

03.474 INF.mk2-values: dc V: 26.4 V
03.474 INF.mk2-values: dc I: 0.0 A
03.474 INF.mk2-values: output frequency 50.0 Hz

==== AC values ====

AC values need to be scaled with the scales obtained above and multiplied by the factors inside the message:

```
out->inputV = mk2UnpackRamFloat(raw->inputV, &dev->ramInfo[WM_VAR_UMAINS_RMS]);
out->inputI = mk2UnpackRamFloat(raw->inputI, &dev->ramInfo[WM_VAR_IMAINS_RMS]) * raw->backfeedFactor;
out->inverterV = mk2UnpackRamFloat(raw->inverterV, &dev->ramInfo[WM_VAR_UINVERTER_RMS]);
out->inverterI = mk2UnpackRamFloat(raw->inverterI, &dev->ramInfo[WM_VAR_IINVERTER_RMS]) * raw->inverterFactor;

/* convert period to frequency in Hz */
out->frequence = mk2UnpackRamFloat(raw->inputPeriod, &dev->ramInfo[WM_VAR_MAINS_PERIOD_TIME]);
if (out->frequence)
        out->frequence = 10 / out->frequence;
```

```
03.568 INF.:   > 03 FF F 01
03.692 INF.:   < 0F 20 01 01 81 79 08 37 53 49 00 37 53 27 00 C3
03.692 INF.mk2-values:    input V: 213.0 V
03.692 INF.mk2-values:    input I: 0.7 A
03.692 INF.mk2-values:    inverter voltage 213.0 V
03.692 INF.mk2-values:    inverter current 0.4 A
03.692 INF.mk2-values:    input period 50.1 Hz
```

Appendix 3 Revision history

Version	Date	Changes
1	14 Nov 2007	Document created.
1.1	11 Feb 2008	Revised descriptions for the message format, and the ‘A’, ‘L’, and ‘S’ commands. Added a section on identifying the protocol in use. Added details of the ‘F’ command.
1.2	16 May 2008	Added details of ‘W’ response concatenation.
1.3	03 July 2008	Added a note about not ending frames with 0xFF. Corrected the description of how the shore current set point is interpreted when the panel scale is set to 255.
1.4	10 July 2008	Added more detail to the description of the info frame response.
1.5	16 July 2008	Added more detail for the MasterMultiLED frame description.
1.6	2 October 2008	Added information on receiving panel frames.
1.7	9 October 2008	Added more detail to the panel frame description.
1.8	26 March 2009	Added example messages.
1.9	20 May 2009	Corrected the DC info example. Re-formatted the examples to reduce endian ambiguity.
2.0	14 September 2009	Deleted “bits” numbered 8-10 in the table describing the <AC Input Configuration> byte of the MasterMultiLED frame.
3.0	04 November 2009	Merged with “VE - interfacing with the Phoenix product range”. Added an example showing the format of frames with appended LED status.
3.1	05 July 2011	Corrected checksum of first example in Appendix 1 Changed footnotes to end notes. Added endnote a)
3.2	18 August 2011	Changed part about zero padding in paragraph 3.2 Inserted paragraph 3.3
3.3	29 August 2011	Changed paragraph 3.4. Explained the <mode> field in more detail.
3.4	28 February 2012	Added DTR info to paragraph 3.1
3.5	19 March 2012	Corrected “Request the DC info frame” example.
3.6	26 March 2012	Added information on MK2 Jumpers, paragraph 3.5 Added information on MK2 powerup sequence Added Appendix 2, Annotated example for typical UI
3.7	18 October 2012	Added note on MK2 firmware 1130132 to chapter 2 Made aware of 26yyzzz and 27yyzzz firmwares
3.8	14 December 2012	Changed chapter 5, added variant 2for sending input current limit, added some notes on input current. Changed ‘shore current’ to ‘input current’ throughout the document.
3.9	13 March 2014	Corrected an erroneous hyperlink (which inserted a complete paragraph in a table) changed Endnotes to Footnotes
3.10	27 January 2015	Added link to data communication whitepaper in the introduction Added instruction in chapter 3 to always make sure the MK2 has the latest firmware in the introduction. Added ‘MK2 protocol’ to the document name.
3.11	17 April 2015	Added last bullet in the final notes in §3.2 Added ‘Request the State of Charge value’ example.
3.12	19 February 2016	Remarks about zero padding are now ‘strikethrough’. Updating to latest MK2 firmware suggested and explained.
3.13	06 June 2017	Added Long Winmon frame support Added XYZ Winmon commands Updated Added CommandWriteViaID Added info about snapshot mode Added Ram variables 14 through 19 Updated CommandSendSoftwareVersion

3.14	22 June 2017	Updated CommandGetSetDeviceState Updated CommandReadRAMVar Updated CommandGetSettingInfo Updated CommandGetRAMVarInfo Fixed target software version example Added reset VE.Bus Devices Added Battery Operational Limits