OBDX Pro VT Command Set Reference Guide

Table of Contents

This page has been left blank intentionally

1. About the OBDX Pro

1.1. What OBD2 Protocols does this tool support?

The OBDX Pro VT supports GM's J1850 VPW protocol. This includes 4x mode and blocks up to 4096bytes (USB only).

1.2. What are the supported developer diagnostic APIs?

The OBDX Pro VT scantool supports two protocols:

- ELM Commands including extended OBDX Commands – String Based Protocol
- DVI (Direct Vehicle Interface) Commands – Byte Based Protocol

The OBDX also supports J2534 (v4.04) using our supplied J2534 DLL (NOT COMPLETED YET). Internally the J2534 DLL uses the DVI protocol for fast and efficient communication.

By default the OBDX starts up with the ELM command set so that it works with most common OBD2 applications which support the ELM protocol.

1.3. Which API is best for development?

If you are new to programming, then the ELM command set uses strings which are easy to understand, and can be easily tested using free applications such as Putty.

The ELM command set supports both legacy ELM commands along with our extended DX commands to allow custom capabilities including advanced filters, passthrough mode and 4x enable.

Our byte based protocol, DVI, is designed to provide the fastest and most efficient communication between the vehicle and software. This protocol is suitable for experienced programmers that are familiar with bits, bytes and arrays.

Examples of using both APIs along with every supported function is provided in this document.

2. ELM Protocol

2.1. Introduction

The ELM protocol is a simple string based command set which allows easy communication between a PC/Mobile using a simple serial application such as Putty. This protocol is extremely easy to get started with, and is great for both beginners and advanced users for learning and connecting to a vehicle.

This tool supports the main legacy ELM command set plus our own extended command set for enhanced capabilities. All ELM commands start with AT, and all OBDX commands start with DX.

Any commands sent that do not start with AT or DX are treated as network frames (Messages sent to the vehicle) and must be a hexadecimal value such as 01 or FA. Hexadecimal is a value between 00 and FF (Google hexadecimal for a quick explanation!), attempting to use any other string characters will result in the scantool returning an Error.

All commands must be terminated with a carriage return (CR), a 'newline'(NL) after the carriage return is accepted also but a CR must be sent.

After every command, the scantool will indicate it is idle by returning the character >. Once received, the next command can be sent and processed.

Example:
ATI

ELM327 V1.4

>

2.2. List of all ELM Commands

Below is a list of all commands implemented on the scantool, please take careful note of any commands indicated compatibility as these are only there for ELM compliancy and are not actually implemented.

Supported ELM AT commands include:

AT@1	ATI	ATSP
ATAR	ATL0/1	ATSPA
ATAT0/1/2	ATMA	ATSR
ATD	ATMR/ATMT	ATST0/1/2
ATDP	ATR0/1	ATSH
ATDPN	ATRV	ATTP
ATE0/1	ATRA	ATTPA
ATH0/1	ATVPW1/4	ATZ
ATL0/1		

Included commands for Compatibility only (Responds 00 or OK)

ATAL	ATCEA	ATCF
ATAMC	ATCRA	ATCM
ATAMT	ATCAF	ATD0/1
ATBD	ATCFC	ATDM1
ATBI	ATCP	ATFCSD
ATBRD	ATCSM	ATFE
ATBRT	ATCTM	ATFI
ATCS	ATCRV	ATFCSM
ATIGN	ATJHF	ATNL
ATIB	ATJTM	ATPC
ATIFR	ATKW	ATPPS
ATIIA	ATLP	ATPB
ATJE	ATM0/1	ATPP
ATJS	ATMP	ATRD
ATRTR	ATSW	ATWS

ATSI	ATTA	ATWM
ATSS	ATV0/1	

Supported OBDX DX commands include:

DP0/1		
SD		
VS1/4		
PT0/1		

For detailed explanation and usage, please see the specific command in this guide.

2.3. AT @1 – Identify Device

This command simply identified the device, the scantool will report: OBDX Pro VT

This command is ideal for identifying the type of OBDX Pro scantool connected.

2.4. AT AR – Reset Filters

Will reset all filters and masks back to default. This will revert to a VPW header of 68 6A 6B

2.5. AT AT – Adaptive Timing

This is enabled by default, and determines how long the scantool will wait for a reply from the vehicle before timing out. A higher value results in a shorter timeout meaning it can perform faster diagnostics, although can miss slow VPW responses. Adaptive timing works out how long it should wait based on the response times of previous message responses.

2.6. AT D – Reset all to Default

Resets all settings back to default

2.7. AT DP – Display Protocol

Displays the current protocol name as a string

2.8. AT DPN - Display Protocol Number

Displays protocol number. VPW only supported thus reports 02

2.9. AT E – Echo

Sets whether user commands are echo'd back. By default this is ON, this can be turned off it wanting to maximise speed between scantool and software. It is recommended to have it on if using a serial terminal such as Putty.
E0 = Echo off
E1 = Echo on

2.10. AT H – Headers

Headers are the first 3 bytes of the VPW frame which indicate the priority, destination and source of the message. This option is OFF by default but is recommended to be turned ON. When enabling headers, this also enables the CRC to be displayed which is the final byte of the message.

2.11. AT I – Identify ELM Legacy Device

The Identify command will report back ELM327 v2.1 for legacy support with other mobile/pc softwares. Some applications require seeing the ELM text before successfully connecting to the cable.

2.12. AT L – Linefeed

The line feed option is enabled by default, this adds a linefeed between carriage returns to space data out further. This is highly recommended to be enabled if using a serial console.
L0 = Linefeed off
L1 = Linefeed on

2.13. AT MA/MR/MT – Monitor All,Receive,Transmit

The monitor commands allow either monitoring all traffic being received, or to monitor based on the destination (Receive) or source (Transmit) of the header bytes.

Once the monitor mode is enabled, you cannot send data to the car as it's a read only mode. As soon as any data is sent to the device, it will exit the monitor mode and return STOPPED.

2.14. AT R – Responses

Responses is turned on by default, Responses command allows information the tool whether it is expecting a response back from the VPW network. For example, if turning off responses, you are able to send a message such as 20, and the tool will instantly send it and return a '>' to indicate idle.

With responses turned on, the scantool will attempt to search for a response frame form the VPW line.

2.15. AT RV – Read Voltage

This command reads the battery voltage on pin16. It is accurate to .1 volts between 5 to 20volts.

An example readout using RV is 14.5v

2.16. AT VPW – VPW Speed

This command sets the VPW speed as 1x (10.4kbps) or 4x (41.6kpbs). The 4x is only required for high speed flash programming ecus, it should not be required for standard usage.

2.17. AT SP – Set Protocol

This will set the cables OBD2 protocol. Only protocol VPW (02) is supported on this scantool

2.18. AT SPA – Set Protocol Automatic

This does exactly the same as SP as there is only one protocol available.

2.19. AT SR – Set Receiver

This command sets a custom filter on the destination (receive) byte. The filter is usually set based on the header bytes set using SH (Set header).

2.20. AT ST – Set Timeout

The timeout command directly controls how long the scantool will wait for responses. The time is calculated in milliseconds using the following calculation: 4x hh = timeout, where hh is the value passed to the command in hexadecimal (ST45).

2.21. AT SH – Set Header

This command will set the VPWs header frame bytes and is also used to determine the filter based on the source byte.

2.22. AT S0/1 – Spaces

This command will enable or disable if spaces are sent between bytes returned by the scantool.

2.23. AT TP – Try Protocol

Does exactly the same as Set Protocol.

2.24. AT TPA – Try Protocol Automatic

Does exactly the same as Set Protocol.

2.25. AT Z – Reset

Does a full reset of the scantool.

2.26. DX DP0/1 – API Diagnostic Protocol

This command allows setting the diagnostic protocol to either ELM (0) or DVI (1) commands.

Once switched to DVI, you must use the byte protocol from there onwards. This command will return OK before switching to the byte protocol

If wanting to switch back to ELM commands, you must use the DVI command for switching back.

2.27. DX I – OBDX Identifier

This command is the same as AT@1 which returns the same id.

2.28. DX SD – Send Data

This command allows sending a full VPW frame including header in on message. It also automatically sets the filter based on the header passed in.

Example: 6C 10 F1 01 0C

2.29. DX VS1/4 – Set VPW Speed between 1x and 4x

Exactly the same command as VPW1/4.

2.30. DX PT0/1 – Passthrough

The passthrough command enabled a unique functionality where the device will return any VPW message received at anytime that matches the filter requirements. This removes the “Send 1, receive 1” nature of the ELM commands so that multiple responses from the VPW line can be returned easily.

2.31. DX SM – Set Mask

This will set the overall filter mask which is applied to both the filter set and frame received to allow customizing which frames are received. By default, the mask is always on for the ELM protocol and is set to FF FF FF.

Example: DX SM FF FF FF

This example will set the mask to FF FF FF which means every bit of the header must match the filter.

Whereas doing DX SM 00 00 00 will mean none of the filter bits will need to match for the frame to pass filter check.

2.32. DX RA – Read Analog

This function is used to read from one the analog pins of the scantool.

DX RA 1

This will respond with a 4 digit hex value such as 015A which when converted to decimal is 346.

2.33. DX RE – Read Errors

This function will read out the current error count for the VPW line.

DX RE 01 or DX RE A

Note, must use HEX for requesting fault codes over 9.

The list of errors to view are:
ReadBusInactive = 0
ReadSOFLongerThenMax = 1
ReadSOFShorterThenMin = 2
Read2BytesOrLess = 3
ReadCRCIncorrect = 4
ReadFilterTOdoesNotMatch = 5
ReadFilterFROMdoesNotMatch = 6
ReadRangeFilterTOdoesNotMatch = 7
ReadRangeFilterFROMdoesNotMatch = 8
WriteFrame_IdleFindTimeout = 9
ReadFrame_NotEnabled = A
NotEnabled = B
READONLYMode = C
ReadFrame_AwaitingSendPC = D

2.34. DX US – Unique Serial

This will return a unique 12byte serial unique to every single scantool.

This is ideal for developers wanting to tie software licensing to a cable.

2.35. Examples

Only 2 main frames need to be send to begin communicating to the car, these are as follows:

AT SP 2 - Set protocol to 2 (VPW)

AT SH 6C10F1 - Set header bytes to send data to 10 (PCM) and from F1 (scantool)

Once done sending network frames is very easy, you simply need to send the desired data bytes and the scantool will display any responses that meet the filter:

01 0C

The above frame is requesting RPM from the engine computer (PCM), an example response could be:

41 0C 02 20

3. OBDX DVI Protocol

3.1. Introduction

This protocol is designed to provide fast and efficient communication to the scantool. This protocol does not use strings like the ELM and requires sending raw bytes over the serial port to the scantool for communication, this means applications such as putty are not suitable for testing and development.

All example commands shown are raw byte format and must not be sent as a string like done with the ELM commands.

To enable using the DVI protocol, you must first send the ELM string command DX DP 1 which will change from ELM to DVI protocol.

3.2. Frame Format including Responses and Faults

The OBDX DVI protocol is a byte based communication, it has 4 core parts to the byte frame which include the following: Command, Frame length, Data and Checksum.

For example:
Request: **22 01 01 YY**
Response: 32 03 01 SS TT YY

Where:
22 = Command
01 = Length of bytes to come (Only includes data bytes)
01 = Data byte (In this case, this is a sub command)
YY = Checksum (Sum of bytes and inversed)

The response from the cable shows:
32 = command plus 0x10
03 = length
01 = sub command
SS TT = data bytes
YY = Checksum

The cable also picks up and display various errors if specific conditions are not met such as incorrect checksum, missing bytes ect.
And example is:
7F 02 22 05 YY

Where:
7F = Error
02 = Length of bytes to come
22 = Command
05 = Error code
YY = Checksum

The above is applicable for all configuration errors. There is a different frame for network frames for both read and writing. Only errors that are requested in the error bits parameter will be displayed.

An example of an error occurring during writing is as followings
PC Sends: 10 06 00 00 07 E0 01 20 YY (Send CAN frame 7E0 01 20)
If successful, Cable responds: 20 01 01 YY

If unsuccessful, a fault is reported as seen below:
7F 02 10 XX YY
where 7F indicates error, 10 is the command, XX is error and YY is checksum

This can include configuration related errors such as invalid format, too long or also network write errors such as bus timing problems ect.

Reading a network frame will display as follows:
08 06 00 00 07 E0 01 20 YY

A fault while reading will be displayed as:
Valid read: 08 06 00 00 07 E0 01 20 YY
Fault: 7F 02 FF 05 YY
Where 7F indicates error, FF indicates error on reading from network and 05 indicates fault code. Do note that faults while reading from bus are disabled by default and must be enabled by enabling the error bits.

The below section indicates all possible faults for configuration/commands and also protocol specific reading errors.

The configuration errors possible include:
const uint8_t Error_InvalidCommand = 1;
const uint8_t Error_RecvTooLong = 2;
const uint8_t Error_ByteWaitTimeout = 3;
const uint8_t Error_InvalidSerialChksum = 4;
const uint8_t Error_SubCommandIncorrectSize = 5;
const uint8_t Error_InvalidSubCommand = 6;
const uint8_t Error_SubCommandInvalidData = 7;

Each protocol has its own predefined list of errors. All errors for each protocol are defined below:

VPW Error Codes
Reading Errors:
ReadBusInactive = 0;
ReadSOFLongerThenMax = 1;
ReadSOFShorterThenMin = 2;
Read2BytesOrLess = 3;
ReadCRCIncorrect = 4;
ReadFilterTOdoesNotMatch = 5;
ReadFilterFROMdoesNotMatch = 6;
ReadRangeFilterTOdoesNotMatch = 7;
ReadRangeFilterFROMdoesNotMatch = 8;
NotEnabled = 10;
READONLYMode = 11;
ReadFrame_AwaitingSendPC = 12;

Writing Errors:
WriteFrame_IdleFindTimeout = 9
NotEnabled = 10
READONLYMode = 11

3.3. Receive from Network Normal (0x08)

This indicates how frames are received from the bus. Frames are automatically processed and sent to the PC via USB or BT. By default, the network is turned off so no frames are received and all filters are set to off. The protocol is enabled when started, disabled when ended. A separate command is required to enable/disable communication or set to read only.

When changing protocols, receiving/writing is disabled and must be enabled again.

It is recommended to set at least one filter before enabling the network to prevent unwanted frames. To monitor all messages, disable all filters and enable network.

3.3.1. VPW Frame Format Example

Example 1: **08 05 6C 10 F1 01 20 YY**
Frame received is: 6C,10,F1,01,20,XX
XX is the VPW frame checksum. This is DISABLED by default
YY is command checksum

Example 2: **08 12 6C 10 F1 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F YY**
Frame received is: 6C,10,F1,01,02,03,04,05,06,07,08,09,0A,0B,0C,0D,0E,0F
VPW frame checksum is disabled in this example
YY is command checksum

Example 3 with timestamp enabled: **08 EE RR TT ZZ 05 6C 10 F1 01 20 YY**
Frame received is: 6C,10,F1,01,20,XX
EE RR TT ZZ is time stamp in microseconds
XX is the VPW frame checksum. This is DISABLED by default
YY is command checksum

3.4. Receive from Network Large (0x09)

Almost exactly the same as the network normal message except this frame uses two bytes for length.

3.4.1. VPW Frame Format Example

Example 1: **09 00 05 6C 10 F1 01 20 YY**
Frame received is: 6C,10,F1,01,20,XX
XX is the VPW frame checksum. This is ENABLED by default
YY is command checksum

Example 2: **09 00 12 6C 10 F1 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F YY**
Frame received is: 6C,10,F1,01,02,03,04,05,06,07,08,09,0A,0B,0C,0D,0E,0F
VPW frame checksum is disabled in this example
YY is command checksum

Example 3 with timestamp enabled: **08 EE RR TT ZZ 00 05 6C 10 F1 01 20 YY**
Frame received is: 6C,10,F1,01,20,XX
EE RR TT ZZ is time stamp in microseconds
XX is the VPW frame checksum. This is DISABLED by default
YY is command checksum

3.5. Send to Network Normal (0x10)

This function is to send normal size frames to the network. Max length is 255bytes in total.

3.5.1. VPW Frame Format Example

Example 1: **10 05 6C 10 F1 01 20 YY**
Frame sent is: 6C,10,F1,01,20 - The VPW frame checksum is automatically calculated.
YY is command checksum

Response: 20 01 00 YY

Example 2: **10 12 6C 10 F1 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F YY**
Frame sent is: 6C,10,F1,01,02,03,04,05,06,07,08,09,0A,0B,0C,0D,0E,0F - The VPW frame checksum is automatically calculated.
YY is command checksum

Response: 20 01 00 YY

3.6. Send to Network Large (0x11)

This function is to send normal size frames to the network. Max length is 12000 bytes in one hit, although do note not all OBD protocols support this length (ie. ALDL is a max of 176bytes).

3.6.1. VPW Frame Format Example

Example 1: **11 00 05 6C 10 F1 01 20 YY**
Frame sent is: 6C,10,F1,01,20 - The VPW frame checksum is automatically calculated.
YY is command checksum

Response: 21 01 00 YY

Example 2: **11 00 12 6C 10 F1 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F YY**
Frame sent is: 6C,10,F1,01,02,03,04,05,06,07,08,09,0A,0B,0C,0D,0E,0F - The VPW frame checksum is automatically calculated.
YY is command checksum

Response: 21 01 00 YY

3.7. Scantool Information (0x22)

The purpose of this function is to read information about the scantool including firmware version, model type, unique ID and other cable specific information.

3.7.1. Request Hardware Version (0x01)

Request: **22 01 00 YY**
Response: 32 03 00 SS TT YY
SSTT is the response. Example 00FF = v2.55

3.7.2. Request Firmware Version (0x01)

Request: **22 01 01 YY**
Response: 32 03 01 SS TT YY
SSTT is the response. Example 00FF = v2.55

3.7.3. Request Model (0x02)

Request: **22 01 02 YY**
Response: 32 03 02 SS TT YY
SSTT is the response. Example 0214 = Model is 214

3.7.4. Request Name (0x03)

Request: **22 01 03 YY**
Response: 32 08 03 11 22 33 44 55 66 77 YY
11 to 77 would be the name in ASCII

3.7.5. Request Unique Serial (0x04)

Request: **22 01 04 YY**
Response: 32 0D 04 11 22 33 44 55 66 77 88 11 22 33 44 YY
11 to 44 is the unique serial as a 96bit value.

3.7.6. Supported OBD Protocols (0x05)

Request: **22 01 05 YY**
Response: 32 03 05 XX NN YY
Where XX NN are supported protocols
On XX, bits to set ALDL (0), VPW (1), HSCAN (2), MSCAN (3), HSCAN (4), PWM (5), LIN (5), ISO9 (6)
NN is for future protocols.
YY is checksum

3.7.7. Supported PC Protocols (0x06)

Request: **22 01 06 YY**
Response: 32 02 06 XX YY
On XX, bits are set as USB (0), Bluetooth (1), WIFI (2).

3.8. Commands Format and Settings (0x24)

The purpose of these commands are to allow changing how frames are sent and processed to and from the scantool.

3.8.1. Network Write Responses Status (0x1)

This function enables/disables the automated acknowledge response from the cable. Only ERROR messages are received.
To check Network response status:
Request: **24 01 01 YY**
Response: 34 02 01 XX YY where XX is 00 (Off) or 01 (On default)

To change Network response status:
Request: **24 02 01 NN YY**
Where 00 = OFF, 01 = ON
Response: 34 02 01 NN YY

3.8.2. Configuration Response Status (0x02)

This function enables/disables the automated acknowledge response from the cable. If requesting information, the cable will respond but if changing settings, no response will occur. ERROR messages will also be received.

To check configuration response status:
Request: **24 01 02 YY**
Response: 34 02 02 XX YY where XX is 00 (Off) or 01 (On default)

To change configuration response status:
Request: **24 02 02 NN YY**
Where 00 = OFF, 01 = ON
Response: 34 02 02 NN YY

3.8.3. Timestamp on Received Network Frame (0x03)

This function enables the network frame timestamp which adds a 4byte timestamp which records in microseconds from when the frame is received. This is OFF by default.

To check timestamp status:
Request: **24 01 03 YY**
Response: 34 02 03 XX YY where XX is 00 (Off) or 01

To change timestamp status:
Request: **24 02 03 NN YY**
Where 00 = OFF, 01 = ON
Response: 34 02 03 NN YY

3.9. Reset Back to Boot (0x25)

The purpose of this command is to reset the scantool back to its boot sequence.

3.9.1. Software Reboot

Request: **25 00 YY**

Response: 35 00 YY

Response is sent BEFORE jumping back to boot

3.10. Set Protocol and Communication Status (0x31)

The purpose of this function is to allow setting the protocol and also enabling/disabling protocol communication.

3.10.1. Setting and Requesting the OBD Protocol

Set Protocol: **31 02 01 XX YY**
Where XX is:
00 = ALDL
01 = VPW
02 = HS CAN
03 = MS CAN
04 = GMLAN
05 = PWM (Not Implemented)
06 = LIN (Not implemented)
Response: **41 02 01 XX YY**
YY is normal checksum

Request Protocol: **31 01 01 YY**
Response: **41 02 01 XX YY**
YY is normal checksum

3.10.2. Enabling and Disabling Network Communication (0x02)

Request: **31 02 02 XX YY**
Where XX is: *OFF by default
00 = OFF, 01 = ON, 02 = LISTEN ONLY
YY is normal checksum
Response: **41 02 02 XX YY**

3.10.3. Change API Diagnostic Protocol (0x06)

Request: **31 02 06 XX YY**
Where XX is:
00 = ELM, 01 = DVI
YY is normal checksum
Response: **41 02 06 XX YY**

3.11. Set VPW Specific Settings (0x33)

The purpose of this function is to perform various VPW specific actions including setting filters, 4x on/off, display CRC on/off, validate CRC ect.

3.11.1. Set/Get To Filter

Request to set filter: **33 03 00 MM XX YY**
Where MM is the filter ID (Eg. 0x10, 0xF1 ect).
And XX is filter status (00 = OFF, 01 = ON)
YY is normal checksum
Response: **43 03 00 MM XX YY**
Request to Get: **33 01 00 YY**

YY is normal checksum
Response: **43 03 00 MM XX YY**

3.11.2. Set/Get From Filter

Request to set: **33 03 01 MM XX YY**
Where MM is the filter ID (Eg. 0xF1, 0x10 ect).
And XX s filter status (00 = OFF, 01 = ON)
YY is normal checksum
Response: **43 03 01 MM XX YY**
Request to Get: **33 01 01 YY**
YY is normal checksum
Response: **43 03 01 MM XX YY**

3.11.3. Set/Get To Range Filter

Request to set filter: **33 04 02 MM NN XX YY**
Where MM is the Start filter ID range (Eg. 0x10).
NN is the End filter ID range (Eg. 0xF1)
And XX is filter status (00 = OFF, 01 = ON)
YY is normal checksum
Response: **43 04 02 MM NN XX YY**
Request to Get: **33 01 02 YY**
YY is normal checksum
Response: **43 04 02 MM NN XX YY**

3.11.4. Set/Get From Range Filter

Request to set filter: **33 04 03 MM NN XX YY**
Where MM is the Start filter ID range (Eg. 0x10).
NN is the End filter ID range (Eg. 0xF1)
And XX is filter status (00 = OFF, 01 = ON)
YY is normal checksum
Response: **43 04 03 MM NN XX YY**
Request to Get: **33 01 03 YY**
YY is normal checksum
Response: **43 04 03 MM NN XX YY**

3.11.5. Set/Get Mask

Request to set: **33 05 04 BB NN MM XX YY**
Where BB NN MM is the Mask ID (Eg FFFFFF).
And XX s mask status (00 = OFF, 01 = ON)
YY is normal checksum
Response: **43 05 04 BB NN MM XX YY**
Request to Get: **33 01 04 YY**
YY is normal checksum
Response: **43 05 04 BB NN MM XX YY**

3.11.6. Set/Get VPW 4x

This is set to OFF by default.
Request to set: **33 02 06 XX YY**
Where XX is: 00= 4x OFF, 01 = 4x ON
YY is normal checksum
Response: **43 02 06 XX YY**
Request to Get: **33 01 06 YY**
YY is normal checksum
Response: **43 02 06 XX YY**

3.11.7. Set/Get Validate CRC on Network Frames

This ON by default, thus CRC's are validated on read.
Request to set: **33 02 07 XX YY**
Where XX is: 00= CRC OFF, 01 = CRC ON
YY is normal checksum
Response: **43 02 07 XX YY**
Request to Get: **33 01 07 YY**
YY is normal checksum
Response: **43 02 07 XX YY**

3.11.8. Set/Get Display CRC in Received Frame

This is OFF by default, thus does not display CRC in received frames to PC.
Request to set: **33 02 08 XX YY**
Where XX is: 00= CRC OFF, 01 = Display CRC in returned frame
YY is normal checksum
Response: **43 02 08 XX YY**
Request to Get: **33 01 08 YY**
YY is normal checksum
Response: **43 02 08 XX YY**

3.11.9. Set/Get Write Idle Timeout Value

This edits the timeout value used for waiting for the network to be idle. Busy bus's may require it to be increased.
Request to set: **33 03 09 NN MM YY**
Where NN MM is the timeout value in MS (Default 500ms)
YY is normal checksum
Response: **43 03 09 NN MM YY**
Request to Get: **33 01 09 YY**
YY is normal checksum
Response: **43 03 09 NN MM YY**

3.11.10. Set/Get 1x VPW Timings

Request to set: **33 04 0A XX NN MM YY**
Where XX is Parameter number
NN MM is 16bit timing value in microseconds
YY is normal checksum
Response: **43 02 0A XX YY**

Request to Get: **33 02 0A XX YY**
Where XX is the Parameter number

YY is normal checksum
Response: **43 04 0A XX NN MM YY**

3.11.11. Set/Get 4x VPW Timings

Request to set: **33 04 0B XX NN MM YY**
Where XX is Parameter number
NN MM is 16bit timing value in microseconds
YY is normal checksum
Response: **43 02 0B XX YY**
Request to Get: **33 01 0B YY**
YY is normal checksum
Response: **43 04 0B XX NN MM YY**

3.11.12. Reset Timings

This will set the timings (1x and 4x) VPW timings to the original values as found at startup.
Request to Set: **33 02 0C XX YY**
Where 00 is 1x
And 01 is 4x
YY is normal checksum
Response: **43 02 0C XX YY**

3.11.13. Set/Get VPW Error Bits

Request to set: **33 03 0D XX NN YY**
Where XX NN is 16bits, each bit is a fault. Default is all network frame receive faults off.
YY is normal checksum
Response: **43 03 0D XX NN YY**
Request to Get: **33 01 0D YY**
YY is normal checksum
Response: **43 03 0D XX NN YY**

3.11.14. Set/Get VPW Error Count

Request to Get: **33 02 0E XX YY**
XX is error number (0 to 13)
Reading the error count will reset it back to 0
YY is normal checksum
Response: **43 03 0E XX NN YY**
XX is bit number, NN is the count

3.11.15. Reset to Default Settings

This function will set all VPW settings to their default values
Request to set: **33 01 0F YY**
YY is normal checksum
Response: **43 01 0F YY**

3.12. ADC Commands (0x3A)

The purpose of these commands are to interact with the onboard analog pins which includes reading battery voltage or other internal or external ADC pins.

Some boards will have external ADC connectors which will be suitable for monitoring sensors.

3.12.1. Request Single ADC Channel (0x0)

This will allow reading 1 ADC channel.

Request to Get: **3A 02 00 XX YY**
XX is the channel (0 to 3 currently);
YY is normal checksum
Response: **4A 04 00 XX NN MM YY**
XX is the requested channel
NN MM is the raw channel response as a 16bit value.

3.12.2. Request multiple ADC Channel (0x1)

This will allow reading multiple ADC channels in a single request.

Request to Get: **3A 02 01 XX YY**
XX is each channel as bits (ie. Bit0 = chan0, bit1=chan1,bit2=chan2,bit3=chan3)
YY is normal checksum

Response: **4A LL 01 XX 11 22 33 44 55 66 77 88 YY**
LL is the length, this will vary depending on how many channels are requested
XX is the channels requested
11 22 would be channel 0 response.
33 44 would be channel 1 response.
55 66 would be channel 2 response.
77 88 would be channel 3 response.