Orion Measurement Solutions Inc.
- A Subsidiary of Freese Enterprises -

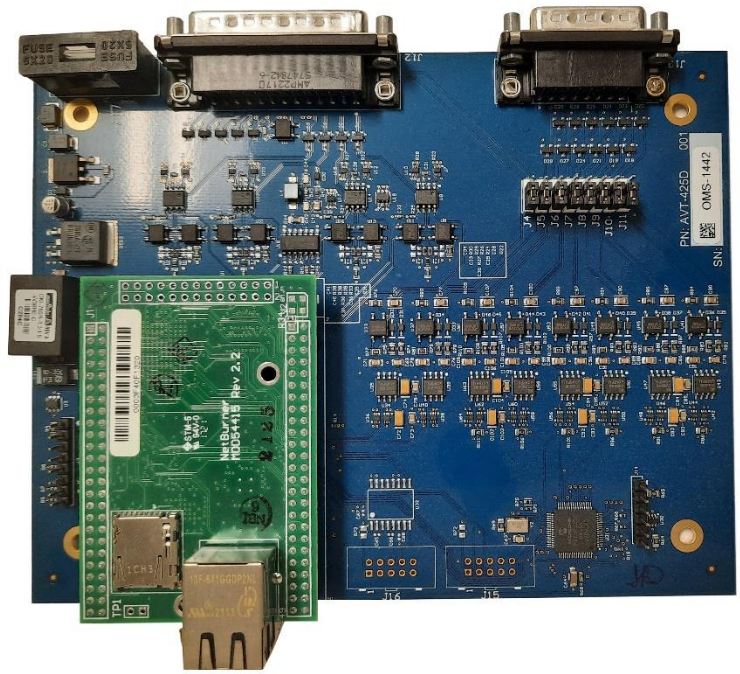
2704 Paldan Dr
Auburn Hills, MI 48326
248-475-9880
sales@orionmeas.com

Manual Covers:

- Hardware: AVT-425 Rev: D, E
- Firmware: Netburner: v50012
- Firmware: ATSAMV70Q19B: v0024
- Firmware: PIC24FJ256GB406: v1007

Reference Documents:

- OMS AVT-425 Datasheet
- OMS AVT-425 Diag Tool User's Manual



Revision Control

Rev Date	Rev	Updated by	Description
10/24/2022	A	Josh Dunn	Initial Revision
6/7/2023	B	Josh Dunn	Added CRC, Rolling Counter, Serial Number support
6/18/2024	C	Don Davis	Changed 2704 Paldan address
7/12/2024	D	Josh Dunn	Added DWN file examples, updated circuit info, updated error responses

Table of Contents

1 Introduction

This document describes the AVT-425 hardware and firmware.
The AVT-425 is a multiple network interface for in-vehicle networks. The operation firmware supports the following networks/protocols on the indicated channels:

- CAN0: Classical CAN; 2-wire.
  channel 0.
- CAN1: Classical CAN; 2-wire or Single Wire CAN (SWC).
  channel 1.
- CAN2: Classical CAN or CAN-FD; 2-wire CAN.
  channel 2.
- CAN3: Classical CAN or CAN-FD.
  channel 3.
- LIN0: LIN only.
  channel 7.
- LIN1. LIN only.
  channel 5.
- KWP0: Keyword Protocol (ISO 14230) only.
  channel 8.
- KWP1: Keyword Protocol (ISO 14230) only.
  channel 6.
- LIN2 thru 7: LIN only.
  channels $A thru $F

All operations are simultaneous with the following exceptions.
LIN0 and KWP0 are mutually exclusive – they share one hardware channel.
LIN1 and KWP1 are mutually exclusive – they share one hardware channel.

1.1 AVT Diag Tool Examples

The .dwn file examples referenced in this manual are part of the AVT Diag Tool. When the AVT Diag Tool runs for the first time, it will create all the .dwn files in the following folder:

C:\Program Files\Orion Measurement Solutions\AVT Diag Tool\AVTdwnFiles

All .dwn file names in this manual act as links to that folder.

A .dwn file is a text file that contains AVT-425 protocol commands that have been set up to use with the AVT Diag Tool application. A .dwn file has the following format:

- Lines that start with a semi-colon (;) are comments and are not transmitted to the AVT-425.
- Lines that are empty are ignored.
- Protocol Lines have the format: hh hh hh … where hh is a Hex number from 00-FF.

The AVT Diag Tool application created by Orion Measurement Solutions (or the AVT Hex Terminal program created by Advanced Vehicle Technologies) will transmit the byte packets of the .dwn file to the AVT-425 as-is, in the order listed in the file. Refer to the AVT Diag Tool user manual for more information.

1.2 Green Notes

'Green' text will indicate changes from the previous version of the manual.

1.3 New Content

Updated for Version 5.0.0.12, which includes:

- Rolling Counter for Periodic Message commands: 7x 40
- Checksum for Periodic Message commands: 7x 41
- Query Serial Number command: 52 81 01

1.4 Hardware

Contact Orion Measurement Solutions with any questions about hardware revision status.

1.4.1 Summary of Hardware Updates

Initial production release is hardware revision "D".

1.5 Firmware Capabilities

At this time, the AVT-425 firmware supports the following capabilities.

1.5.1 CAN0

CAN (so-called Classical or non-FD). Transmit and receive. Sixteen objects (either receive or transmit). Sixty-four (64 decimal) periodic messages. Periodic messages are Type1 only.

1.5.2 CAN1

CAN (so-called Classical or non-FD). Transmit and receive. Sixteen objects (either receive or transmit). Sixty-four (64 decimal) periodic messages. Periodic messages are Type1 only.

1.5.3 CAN2

Classical CAN and CAN-FD capable. Transmit and receive. Sixteen receive objects. Sixteen transmit objects. Sixty-four (64 decimal) periodic messages. Periodic messages are Type1 only. ISO CAN frame CRC (non-ISO possibly available). Maximum data payload of 64 bytes supported. Maximum baud rate of 8 Mbaud supported.

1.5.4 CAN3

Classical CAN and CAN-FD capable. Transmit and receive. Sixteen receive objects. Sixteen transmit objects. Sixty-four (64 decimal) periodic messages. Periodic messages are Type1 only. ISO CAN frame CRC (non-ISO possibly available). Maximum data payload of 64 bytes supported. Maximum baud rate of 8 Mbaud supported.

1.5.5 LIN0 or KWP0

LIN or KWP operations including receive, transmit, and sixteen (16 decimal) periodic messages.

1.5.6 LIN1 or KWP1

LIN or KWP operations including receive, transmit, and sixteen (16 decimal) periodic messages.

1.5.7 LIN2 thru LIN7

Six additional channels of LIN only operations including receive, transmit, and sixteen (16 decimal) periodic messages.

1.6 Firmware Versions

AVT-425 firmware revisions are available on the Orion Measurement Solutions website.
https://orionmeas.com/

Contact OMS support about additional questions for the AVT-425

1.6.1 Determining Firmware Version

Perform the following to determine the version of firmware in your unit.

- Power-on the AVT-425 interface unit.
- Connect to a Client computer running the AVT Diag Tool or equivalent.
- The connect notification is:
- $91 $0A indicates first connection after power-on.
- $91 $3A indicates AVT-425 operations.
- $93 $04 $xx $yy where 'xx yy' is the firmware version.
- At any time, send the $B1 01 command ('Request Netburner firmware version number' from the 'Diagnostic Message' drop down of the AVT Diag Tool).
- The response will be: $93 $04 $xx $yy where 'xx yy' is the firmware version.

1.6.2 Determining Model Number

Perform the following to determine the model number of your hardware.

- Power on the AVT-425 interface unit.
- Connect to a Client computer running the AVT Diag Tool or equivalent.
- The connect notification is:
- $91 $0A indicates first connection after power-on.
- $91 $3A indicates AVT-425 operations.
- $93 $04 $xx $yy where 'xx yy' is the firmware version.
- Send the 'B1 03' command ('Request model number' from the 'Diagnostic Message' drop down of the AVT Diag Tool).
- The response will be: $93 $28 $xx $yy where xxyy forms the model number. (eg. 04 25)

1.6.3 Determining Board Revision Level

The board revision level is noted on the top (component) side of the board, in white silkscreen, just above the serial number block.

Commands and Responses

A list of commands, responses, and error codes is provided at the end of this document.

2 Glossary

Common terms, abbreviations, acronyms, and more.

$1234	Indicates hex number 1234.
0x1234	Indicates hex number 1234.
	(This manual will not typically use this format as it can be confused with other uses of '0x'.)
BRS	A CAN bit.
	Bit Rate Switch. Part of CAN-FD format.
	Signals that the data portion of the CAN frame uses the data portion bit rate.
CAN	Controller Area Network.
CAN-FD	CAN with "Flexible Data".
	There are two components to CAN-FD.
	Larger data payload; maximum of 64 bytes and
	higher baud rate for the data portion of the CAN frame.
CAN0	CAN, channel 0
CAN1	CAN, channel 1
CAN2	CAN, channel 2
CAN3	CAN, channel 3
FDF	A CAN bit.
	FD Format. Generally meaning it may have a larger data payload than a Classical-CAN frame.
	Part of CAN-FD. Signals that the CAN frame is "FD" formatted.
IDE	A CAN bit.
	ID Extended.
	When this bit = 0 the CAN frame uses an 11-bit ID.
	When this bit = 1 the CAN frame uses a 29-bit ID; extended ID.
ISO 11898-2	ISO specification for high speed 2-wire CAN physical layer.
ISO 15765	An ISO specification dealing with the formatting of data in the CAN frame data field. Also used in sending blocks of data using CAN. Also known as Multi-Frame Messaging (MFM) or Segmented Messages. This specification also deals with other CAN network issues.
J2411	An SAE specification for Single Wire CAN (SWC).
K-line	Single wire communications protocol. Refer to ISO 9141, ISO 9141-2, and ISO 14230 for more information.
KWP	Key Word Protocol. Several versions exist, the most common being Key Word Protocol 2000, which is ISO 14230. This is a 'superset' of ISO 9141 and ISO 9141-2.
KWP0	KWP, channel 8. Uses pin # 23.
KWP1	KWP, channel 6. Uses pin # 11.
LIN	Local Interconnect Network.
LIN0	LIN, channel 7. Uses pin # 23.
LIN1	LIN, channel 5. Uses pin # 11.

LIN2 thru LIN7: LIN, channels $A thru $F.

RTR A CAN bit.
Remote Transmission Request.
When this bit = 1 it indicates a frame that is requesting a remote node to transmit an answering frame.

SRR A CAN bit.
Substitute Remote Request.
A fixed recessive bit that only exists in extended frames (IDE = 1, 29-bit ID).

TVS Transient Voltage Suppression.

Type1 Type1 Periodic Message, CAN, each periodic message operates independently.

Type2 Type2 Periodic Message, CAN, messages operate sequentially.

SWC Single Wire CAN (SAE J2411).

XOR Bit-wise logical exclusive OR.

3 AVT-425 Operation

The AVT-425 does not have a power switch. The unit powers up and begins operations as soon as external power is applied.

Note that the Client computer cannot establish a TCP/IP connection until the AVT-425 is fully operational. From power-on to full operation is about 3 seconds.

4 AVT-425 CPU

The AVT-425 uses a "Netburner Mod 54415-100" CPU module with the following:

- 32-bit, 250 MHz, NXP/Freescale Coldfire processor.
- 64 Mbytes of RAM.
- 32 Mbytes of FLASH.

4.1 Support Software

AutoUpdate and OMSIPSetup.exe are PC applications provided by Netburner.

The AVT Diag Tool installation includes them as part of its installation. There are 2 buttons on the GUI of the AVT Diag Tool that will launch them.

4.1.1 OMSIPSetup

The 'OMSIPSetup' application will attempt to find all Netburner hardware, display the IP address and allow you to view and change the IP address and the subnet mask.

NOTE: If your computer (the Client) is on a different subnet than the AVT-425, this application will likely not be able to find it. Temporarily change the subnet mask and/or IP address of the Client computer to something in the same domain as the AVT-425.

For example, the factory default IP address of the AVT-425 is 192.168.1.70.
If your computer has an IP address that is NOT of the form 192.168.1.xxx then it's likely you will NOT be able to find the AVT-425. Temporarily change the IP address of your computer to something of the

form: 192.168.1.xxx (but not 70) and then run the 'IPSetup' application. When done, return the IP address of your computer to its original setting.

4.1.2 AutoUpdate

The 'AutoUpdate' application will allow you to update the AVT-425 operation firmware for the Netburner processor.

You can use the 'Find…' feature of the 'AutoUpdate' application to find the unit you want to update, or you can enter the IP address manually.

You will need the new AVT-425 operation firmware file. Obtain the file from Orion Measurement Solutions.

5 Client Computer Connection

The AVT-425 is an Ethernet TCP/IP server.
The user or test computer is, therefore, a client.

5.1 AVT-425 Connection to Client Computer

5.1.1 Connection Settings

5.1.1.1 Ethernet IP Address

The factory default IP address of the AVT-425 is static and is set to:
192.168.1.70

The factory default net mask setting is:
255.255.255.0

Depending on the particular network environment in which the AVT-425 is being used, the setting of the net mask may not be important. Rule of thumb: if connected to a busy network set the net mask to 255.255.255.0.

5.1.1.2 Hardware or MAC Address

The AVT-425 uses a Netburner MOD54415 CPU module. The MAC address of that module is indicated on a sticker on the top of the module.

You can send the 'B1 04' command to request the MAC address ('Request MAC address' from the 'Diagnostic Message' drop down of the AVT Diag Tool).

You can also obtain the MAC address from the Client computer ARP table. One way this can be done is to connect the AVT-425 to the network. From the Client computer, open a command window. Ping the AVT-425 using the command: "ping 192.168.1.70". Then query the ARP table using the command "arp –a". The ARP table will show the IP and MAC addresses.

5.1.1.3 TCP/IP Port

Communications with the AVT-425 vehicle network interface is via port # 10001, 10002, 10003, or 10004. Refer to Multiple Client Connections for more information about multiple Client connections.

All communications with the AVT-425 vehicle interface is in binary bytes [not ASCII hex]. Refer to Packet Communications Between the AVT-425 and the Client Computer for a description of the 'packetized' communications protocol between the AVT-425 and the Client computer. All communications with the AVT-425 follow the exact same rules and formats as all other AVT interface equipment.

5.1.2 Ethernet IP Addressing Modes

Two IP addressing modes are available for the AVT-425.

- Static
- DHCP

5.1.2.1 Static IP Addressing

The factory default addressing mode for the AVT-425 is static and the address is set to 192.168.1.70. In static mode the Ethernet address of the AVT-425 is always the same and does not change when power is cycled.

5.1.2.2 DHCP Addressing

Setting the AVT-425 IP address to 0.0.0.0 will enable DHCP (Dynamic Host Configuration Protocol) function.

In this mode, the AVT-425 will, on power-up, search for a DHCP server. If one is found it will obtain its IP address, gateway address, and subnet mask from the DHCP server.

If a DHCP server is not found, the AVT-425 will then assign itself an IP address using the ARP method. The IP address will be of the form 169.xxx.xxx.xxx.

5.1.3 Changing the IP Address

To change or set a static IP address for the AVT-425 you should use the Netburner supplied software; described above in OMSIPSetup.

5.2 Multiple Client Connections

Four simultaneous Client connects are supported.
All four connections are to the same IP address. But they use three different port numbers.
The four available Client connection ports are:
TCP port # 10001.
TCP port # 10002.
TCP port # 10003.
TCP port # 10004.
All four connections operate identically, as described in the next section.

5.2.1 Multi-Client Operation Description

A Client can connect to any available port.
Only one Client connection per port.

When more than one Client is connected to the AVT-425 and when processing commands from a Client - port # 10001 has higher priority than port # 10002. Similarly port # 10002 as higher priority than port # 10003. Specifically, this means that if two or more commands are received at exactly the same time, the command from the lowest numbered port will be processed first. That is all 'higher priority' means in this context. Otherwise, all Clients (ports) are treated equally.

Important Note: All responses are sent out all active ports. This is an important point. For example, if the Client on port # 10002 sends the 'B1 01' firmware version request, the AVT-425 will send the '93 04 xx yy' response out all active ports. That means a Client that did NOT send that command will still receive the response.

An even more important example is: Client #3 sets-up a CAN object to receive certain CAN messages. Those messages, when received, will be sent to all connected Clients.

5.3 Packet Communications Between the AVT-425 and the Client Computer

Communications between the Client computer and the AVT-425, in both directions, uses a 'packet' protocol.

- The first byte of a packet is the header byte.
- The header byte upper nibble (first hex digit) indicates what the packet is about.
- The header byte lower nibble (second hex digit) is the count of bytes to follow.
- If the header byte upper nibble is a zero (0) then the packet is a message to or from the network.
- This protocol is limited to 15 bytes following the header byte (lower nibble = $F).

Some transmit commands and receive responses require more than 15 bytes. For such a situation there are two alternate header formats, which are of the form:
$11 xx
$12 xx yy

These alternate header formats only apply to messages to or from the network.
If the byte count is more than $0F but equal to or less than $FF the packet will be of the form:
$11 xx rr ss tt ...
$11 indicates first alternate header format.
$xx indicates the count of bytes to follow (not including the xx byte).
$rr ss tt ... the packet data, including the message to/from the network.

If the byte count is more than $FF but less than or equal to $FF FF the packet will be of the form:
$12 xx yy rr ss tt
$12 indicates second alternate header format.
$xx yy indicates the count of bytes to follow (not including the xx yy bytes).
$rr ss tt ... the packet data, including the message to/from the network.

For channels CAN2 and CAN3 only:

- The '5x 06' command can be used to force the AVT-425 to always send messages

- received from the CAN bus using only the '12 xx yy' long format.

Example #1

- Turn on the time stamp function for CAN3.
- Command: 53 08 03 01.
- Header byte upper nibble 5 indicates a configuration command.
- Header byte lower nibble 3 indicates three bytes follow.
- $08 is the time stamp command.
- $03 indicates channel 3.
- $01 enable time stamps.

Example #2

- Send a message, to LIN0, as Master, ID = $3C, 8 data bytes.
- Command: 0B 05 01 3C 01 02 03 04 05 06 07 08.
- Header byte = $0B.
- Upper nibble $0 indicates 'to the network'.
- Lower nibble $B indicates 11 bytes follow.
- $01 indicates send as Master.
- $3C is the LIN message ID.
- $01 02 03 … are the 8 data bytes.

Example #3

- Receive a message from CAN3, 11-bit ID, with 12 data bytes.
- Response: $11 10 33 05 07 77 01 02 03 04 05 06 07 08 09 10 11 12
- Header byte: $11, alternate header format #1.
- $10: 16 bytes follow
- $33: FDF and BRS bits are set; channel 03; CAN3.
- $05: receive object 05.
- $07 77: CAN frame ID.
- 01 02 03, … data bytes.

6 Power and Network Connection

The power and network connector (J12) is an industry standard DB-25P connector and requires a DB-25S mate. The pin / signal assignments for the vehicle / network connector are listed here.

Pins with no signal assignment are not connected and should not be used.
The user should not connect anything to those pins.

6.1 AVT-425 DB25 (J12) Connections - Board Revision “D”

Pin #	Description	Notes
1	CAN0_H	
14	CAN0_L	
2	CAN1_H	Transceiver is software selected.
15	CAN1_L	Transceiver is software selected.
3	CAN2_H	
16	CAN2_L	
4	CAN3_H	
17	CAN3_L	
5		
18		
6	SSR-4	Solid state relay output.
19	SSR-3	Solid state relay output.
7		
20		
8	LIN0	Bus power input (supply)
21		
9	LIN1	Bus power input (supply)
22		
10	CAN1_SWC	Transceiver is software selected.
23	LIN0 – (channel 7)	Bus connection.
11	LIN1 – (channel 5)	Bus connection.
24	GND	Only one connection required.
12	GND	Only one connection required.
25	RAW_VIN	Only one connection required.
13	RAW_VIN	Only one connection required.

J12 (the DB-25P connector on the AVT-425 board)
Table 1

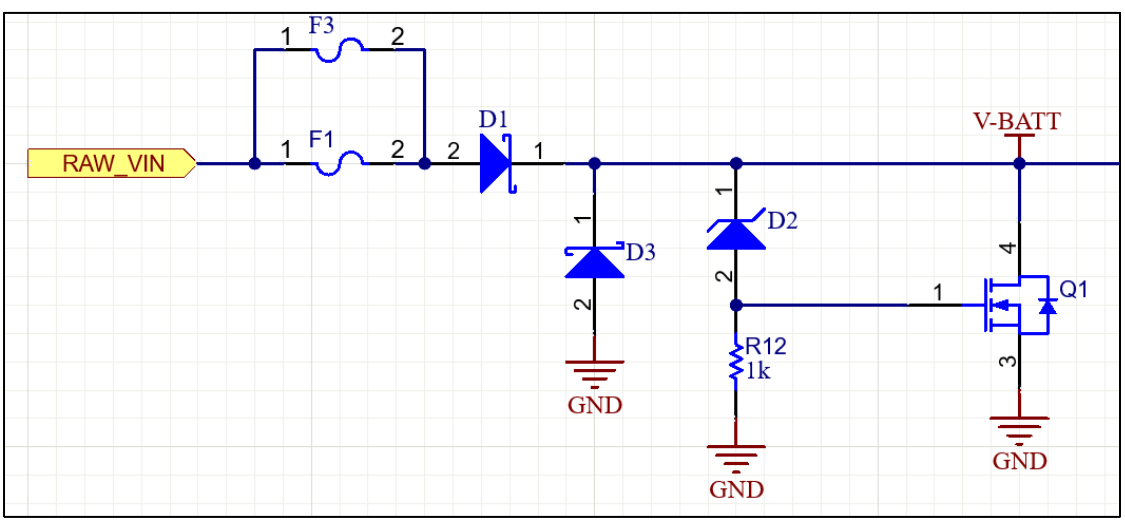
6.2 AVT-425 DB15 (J13) Connections - Board Revision "D"

Pin #	Description	Notes
1	LIN2 – (channel $A)	Bus connection.
9	LIN2	Bus power input (supply)
2	LIN3 – (channel $B)	Bus connection.
10	LIN3	Bus power input (supply)
3	LIN4 – (channel $C)	Bus connection.
11	LIN4	Bus power input (supply)
4	LIN5 – (channel $D)	Bus connection.
12	LIN5	Bus power input (supply)
5	LIN6 – (channel $E)	Bus connection.
13	LIN6	Bus power input (supply)
6	LIN7 – (channel $F)	Bus connection.
14	LIN7	Bus power input (supply)
7		
15	GND.	
8		

J13 (the DA-15P connector on the AVT-425 board)
Table 2

6.3 Power Requirements

The AVT-425 board requires a suitable external power supply. Fairly clean +8 to +18 VDC. The maximum supply voltage is +18 VDC. The voltage input circuit is shown below:



6.3.1 Fuse

The AVT-425 board uses a 5 x 20 mm fuse, 500 mA, fast blow (quick acting) fuse for circuit protection. The factory supplied fuse is Schurter part number: 0034.1516 (Mouser catalog number: 693-0034.1516). Note that the two parallel fuses shown in the diagram represent the two types of fuse configurations that the circuit is designed for. On the final board, only one fuse is populated.

6.3.2 Reverse Voltage Protection

The AVT-425 is protected from reverse voltages by two ON Semiconductor SS24T3G Schottky diodes, one in series with the circuit and the other in parallel (D1 and D3 in the Power Requirements diagram).

6.3.3 Over-Voltage Protection

The AVT-425 contains a crowbar circuit which will blow the fuse in the event of an over-voltage condition. The circuit consists of an ON Semiconductor MMSZ5248CT1G 18V Zener diode, a 1kohm resistor, and an ON Semiconductor NTD3055L104G N-Channel MOSFET (D2, R12, and Q1 in the Power Requirements diagram). The over-voltage threshold is approximately +18 VDC.

6.3.4 Ground

Common ground is required between the AVT-425 board and all connected devices. On J12 there are two 'ground' pins, #12 and #24. Both are connected directly to the ground plane of the AVT-425 board. Only one is needed for normal operations.

6.3.5 Input Voltage

The external power supply is connected to J12 pin #13 or #25. The two pins are connected together, internally, on the AVT-425 board. Only one is needed for normal operations.

6.3.6 Power Dissipation

Power dissipation of the AVT-425 is listed below.
Current draw, minimum, maximum, and average - were measured using a 100 msec sample window. (Fluke Digital Multimeter model 87.) Measurements taken with board connected to Client (logical connection) but no network activity and no activity between the board and the Client.

Input Voltage	Max / Min / Ave Measured Current	Power
+8.0 VDC	306.0 / 296.8 / 302.0 mA	2.42 W
+10.0 VDC	244.8 / 238.0 / 242.0 mA	2.42 W
+12.0 VDC	207.2 / 202.0 / 204.9 mA	2.46 W
+15.0 VDC	171.6 / 167.2 / 169.7 mA	2.55 W
+18.0 VDC	150.0 / 146.4 / 148.4 mA	2.67 W

7 Digital Output

The AVT-425 includes the outputs of a Solid State Relay (SSR).
The SSR is controlled by the AVT-425 CPU by a Client command.

7.1 Digital Output Connection

The digital output is the switched side of the SSR.

It is available on pins # 6 and # 19 of J12 – the DB25P connector on the AVT-425 board.
The SSR is configured such that the user can use it for switching AC or DC voltage and as either a high-side or low-side switch.
The SSR is: IXYS part number "CPC1017N".
A quick overview of SSR ratings:
60 volts maximum blocking voltage
100 mA maximum continuous load current
16 ohm on-resistance, maximum

7.2 Digital Output Operation

The Client controls the digital output using the '5x 05' command.
With the '5x 05' command the Client can send a digital output (rising or falling edge) and synchronize that with the resetting of the time stamp counter.

8 Operation Modes

The AVT-425 interface powers-on with all networks initialized and disabled.

8.1 Simultaneous Network Operations

With three exceptions, all networks can be operated simultaneously.

Exception #1.
CAN1 is selected by the Client to be either a 2-wire or a Single Wire CAN channel.
They are separate transceivers. They cannot operate at the same time.

Exception #2.
LIN0 and KWP0 share a transceiver. Therefore, simultaneous operations for LIN0 and KWP0 are not possible. Pin # 23 on J12 can be operated as LIN0 (channel 7) or KWP0 (channel 8).

Exception #3.
LIN1 and KWP1 share a transceiver. Therefore, simultaneous operations for LIN1 and KWP1 are not possible. Pin # 11 on J12 can be operated as LIN1 (channel 5) or KWP1 (channel 6).

8.2 Start-up Parameters Stored in Non-Volatile Memory

The user has the ability to store some start-up parameters in non-volatile memory. The stored parameters are loaded when the AVT-425 application begins running.

The Client can query for the stored value with the '51 80' command.
The Client can write desired stored values with the '55 80 rr ss tt vv' command.

All the parameters are stored as a bit map in one 32-bit long-word. Unused bits are reserved and should be set to "1".

Note: The '5x 80' command controls the "stored" value used after a reset.

The Client can control the state of any of these states, on-the-fly, by using one of the designated commands listed here. Note: The changes caused by these commands are not persistent. The states will revert to the stored values after each reset.

To control the CAN transceiver (disable/enable) – refer to the '72 13 0r' command.
To control the CAN transceiver termination (out/in) – refer to the '72 62 0r' command.

Bits 07:04 are used to specify the power-on state of the CAN 2-wire transceivers for channels CAN0, CAN1, CAN2, and CAN3. The assignment and states are:

bit 07:	CAN3	0 = transceiver disabled.
		1 = transceiver enabled. [default]
bit 06:	CAN2	0 = transceiver disabled.
		1 = transceiver enabled. [default]
bit 05:	CAN1	0 = transceiver disabled.
		1 = transceiver enabled. [default]
bit 04:	CAN0	0 = transceiver disabled.
		1 = transceiver enabled. [default]

Bits 03:00 are used to specify the power-on state of the CAN bus termination for channels CAN0, CAN1, CAN2, and CAN3. The assignment and states are:

bit 03:	CAN3	0 = termination disabled (not in-circuit).
		1 = termination enabled (in-circuit). [default]
bit 02:	CAN2	0 = termination disabled (not in-circuit).
		1 = termination enabled (in-circuit). [default]
bit 01:	CAN1	0 = termination disabled (not in-circuit).
		1 = termination enabled (in-circuit). [default]
bit 00:	CAN0	0 = termination disabled (not in-circuit).
		1 = termination enabled (in-circuit). [default]

After writing a new stored value, that stored value will not take effect until the next reboot. For example, the user wants CAN1 termination to be disabled and the other channels to be enabled. Store the value 0xFF FF FF FD then reset the application. Do this using these two commands:

55 80 FF FF FF FD	Store the new setting.
F1 A5	Reset the AVT-425 application.

NOTE: This only needs to be done once. The setting is persistent, so it will NOT need to be set again after a reset cycle.

8.3 The Different 'automatic' responses; AVT-425 to Client

The following actions result in the listed responses to be sent to the Client.

Power-on reset or a CPU reset.

  91 0A.

Response to a successful Ethernet connection:

91 3A	operational report.
93 04 xx yy	firmware version report.

Response to an 'F1 A5' reset command.

91 0F	successful reset of the AVT-425 application.

8.4 Types of Resets

The AVT-425 uses the "uCOS" operating system.
On power-up, the operating system then loads and runs the AVT-425 application.

8.4.1 The 'F1 A5' Reset

The 'F1 A5' reset only resets the AVT-425 application. It does not affect the Ethernet connection. Hence, this reset should not cause the Ethernet connection between the AVT-425 and Client to be lost. This reset performs the following functions: It re-initializes all application variables and re-initializes all peripheral hardware; such as the CAN controllers and LIN UART controllers.
The response sent to the Client for the 'F1 A5' reset is: '91 0F'.

8.4.2 The 'F1 C3' Reset

The 'F1 C3' reset causes an operating system reset. This will cause loss of Ethernet connection, re-boot the operating system and then re-load the AVT-425 application. This is very close, but not quite the same as a power reset.
The response sent to the Client for the 'F1 C3' reset is: '91 0A'. (Which will only be received after the Client establishes an Ethernet connection with the AVT-425.)

9 Network Hardware Descriptions

Technical details of each network channel is described in the following sections.

9.1 CAN0 - 2-wire CAN

CAN0 is a high speed 2-wire CAN channel that is ISO 11898-2 compliant.

It uses the Microchip MCP2544FD-H/SN transceiver.

CAN0 is not CAN-FD capable.

Refer to Table 1 for pin and signal definitions.

Termination can be Client selected to be 'in' or 'out'. The default is 'in'. Refer to the '$7x 62' command.

The AVT-425 board has been designed to support several different network termination schemes for CAN0. The factory default is the split termination consisting of two 52.3 ohm resistors in series across the CAN_H and CAN_L signal lines. The mid-point of the two termination resistors is routed through a 10 ohm resistor and a 10,000 pF ceramic capacitor to ground. This configuration provides the standard 120 ohm DC termination and provides good common mode noise rejection. The two opto-relays used to switch the termination add 7 ohms each.

9.1.1 CAN0 Channel Number

For the Client, CAN0 is designated channel 0.
Note: Bits in the upper nibble of the channel number have special meaning for some commands and responses.

9.2 CAN1 - 2-wire CAN or Single Wire CAN

CAN1 is either a high speed 2-wire CAN channel that is ISO 11898-2 compliant or a low speed Single Wire CAN (SWC) channel that is J2411 compliant.

It uses the Microchip MCP2544FD-H/SN transceiver.

For single wire operations it uses the NXP MC33897CTEF transceiver.

CAN1 is not CAN-FD capable.

Refer to Table 1 for pin and signal definitions.

For 2-wire operations, termination can be selected by the Client to be 'in' or 'out'. The default is 'in'. Refer to the '$7x 62' command.

The AVT-425 board has been designed to support several different network termination schemes for CAN1 2-wire operations. The factory default is the split termination consisting of two 52.3 ohm resistors in series across the CAN_H and CAN_L signal lines. The mid-point of the two termination resistors is routed through a 10 ohm resistor and a 10,000 pF ceramic capacitor to ground. This configuration provides the standard 120 ohm DC termination and provides good common mode noise rejection. The two opto-relays used to switch the termination add 7 ohms each.

9.2.1 CAN1 Channel Number

For the Client, CAN1 is designated channel 1.
Note: Bits in the upper nibble of the channel number have special meaning for some commands and responses.

9.3 CAN2 - 2-wire CAN

CAN2 is a high speed 2-wire CAN channel that is ISO 11898-2 compliant.

It uses the Microchip MCP2544FD-H/SN transceiver.

CAN3 is both Classical CAN and CAN-FD capable.

Refer to Table 1 for pin and signal definitions.

Termination can be selected by the Client to be 'in' or 'out'. The default is 'in'. Refer to the '7x 62' command.

The AVT-425 board has been designed to support several different network termination schemes for CAN2. The factory default is the split termination consisting of two 52.3 ohm resistors in series across the CAN_H and CAN_L signal lines. The mid-point of the two termination resistors is routed through a 10 ohm resistor and a 10,000 pF ceramic capacitor to ground. This configuration provides the standard 120 ohm DC termination and provides good common mode noise rejection. The two opto-relays used to switch the termination add 7 ohms each.

9.3.1 CAN2 Channel Number

For the Client, CAN2 is designated channel 2.

Note: Bits in the upper nibble of the channel number have special meaning for some commands and responses.

9.4 CAN3 - 2-wire CAN

CAN3 is a high speed 2-wire CAN channel that is ISO 11898-2 compliant.

It uses the Microchip MCP2544FD-H/SN transceiver.

CAN3 is both Classical CAN and CAN-FD capable.

Refer to Table 1 for pin and signal definitions.

Termination can be selected by the Client to be 'in' or 'out'. The default is 'in'. Refer to the '$7x 62' command.

The AVT-425 board has been designed to support several different network termination schemes for CAN3. The factory default is the split termination consisting of two 52.3 ohm resistors in series across the CAN_H and CAN_L signal lines. The mid-point of the two termination resistors is routed through a 10 ohm resistor and a 10,000 pF ceramic capacitor to ground. This configuration provides the standard 120 ohm DC termination and provides good common mode noise rejection. The two opto-relays used to switch the termination add 7 ohms each.

9.4.1 CAN3 Channel Number

For the Client, CAN3 is designated channel 3.
Note: Bits in the upper nibble of the channel number have special meaning for some commands and responses.

9.5 LIN0

The LIN0 bus is a low speed, single wire, multi-drop, ground referenced network.

The AVT-425 uses the NXP MC33660 transceiver. Maximum baud rate is inferred to be 150 kbps.

The bus (K-line) is pulled up the supply voltage through a resistor. The Client can select between either a 1 K ohm or a 31 K ohm pull-up resistor. Refer to LIN / KWP Bus Supply Voltage for details about the pull-up resistor.

9.5.1 LIN0 Channel Number

LIN0 is designated channel 7.

9.6 LIN1

The LIN1 bus is a low speed, single wire, multi-drop, ground referenced network.

The AVT-425 uses the NXP MC33660 transceiver. Maximum baud rate is inferred to be 150 kbps.

The bus (K-line) is pulled up the supply voltage through a resistor. The Client can select between either a 1 K ohm or a 31 K ohm pull-up resistor. Refer to LIN / KWP Bus Supply Voltage for details about the pull-up resistor.

9.6.1 LIN1 Channel Number

LIN1 is designated channel 5.

9.7 KWP0

Key Word Protocol communications uses the LIN0 transceiver and associated K-line.

Refer to LIN0, above, for information about the physical layer.

Refer to Table 1 for pin number.

9.7.1 KWP0 Channel Number

KWP0 is designated channel 8.

9.8 KWP1

Key Word Protocol communications uses the LIN1 transceiver and associated K-line.

Refer to LIN1, above, for information about the physical layer.

Refer to Table 1 for pin number.

9.8.1 KWP1 Channel Number

KWP1 is designated channel 6.

9.9 LIN2 thru LIN7

If installed, the AVT-425 offers six additional LIN channels.

9.9.1 LIN2 thru LIN7 Channel Numbers

LIN2 thru LIN7 are designated as channels $A thru $F.

10 CAN Channel Operations

A CAN network has to consist of at least two functioning CAN nodes.

Each CAN channel of the AVT-425 is independent of all other channels. This applies to all channel parameters.

10.1 CAN and CAN-FD notes

CAN0 and CAN1 only support Classical CAN formatted frames. This means 11 and 29-bit IDs, 0 to 8 data bytes, RTR are all supported.

If a CAN-FD formatted frame is received by CAN0 or CAN1, the controller will interpret the frame as being in error and will transmit an error frame.

CAN2 and CAN3 support Classical CAN as well as CAN-FD. The Client controls the format of the CAN frame through the FDF bit in the transmit command.

FDF is bit 5 of the object byte in a transmit command.
When the FDF bit is cleared (a '0'), the channel will send a Classical CAN frame.
When the FDF bit is set (a '1'), the channel will send a CAN-FD formatted frame.
BRS is bit 4 of the object byte in a transmit command.
When BRS is cleared (a '0'), the channel will send the data field at the 'slow' baud rate.
When BRS is set (a '1'), the channel will send the data field at the 'fast' baud rate.

If FDF is set, BRS may or may not be set, as the user desires. If BRS is cleared (a '0') , the data portion of the CAN-FD frame is transmitted at the same baud rate as the rest of the frame. Conversely, if BRS is set (a '1') then the data portion of the CAN-FD frame is transmitted at the "fast" baud rate.

If BRS is set, FDF must be set – else the command is in error.

RTR is not supported in CAN-FD. If RTR set, both FDF and BRS must cleared – else the command is in error.

The above statements apply to transmit commands (0x, 11 xx, and 12 xx yy), object and mask configuration commands ('7x 2A' and '7x 2C'), and the periodic message set-up command ('7x 18').

CAN2 and CAN3 can receive both Classical and FD formatted frames without changing configuration. What the Client receives is controlled by the ID and mask configuration; the '7x 2A' and '7x 2C' commands.

10.2 CAN2 and CAN3 - Core Notes

CAN2 and CAN3 are implemented by a Microchip "ATSAMV70Q19B-AABT" microcontroller running proprietary firmware.
The CAN-FD controller in the ATSAM microcontroller is Bosch MCAN IP. The MCAN core version can be obtained using the 'B1 02' command.
The CAN2/3 (ATSAM microcontroller) firmware version can be obtained by using the 'B1 07' command ('Request CAN2/3 firmware version number' from the 'Diagnostic Message' drop down of the AVT Diag Tool).

10.3 CAN Channel Operational Modes

Each CAN channel of the AVT-425 has two operating modes:
Disabled
Normal

10.3.1 Disabled

The CAN channel cannot receive any messages and it cannot transmit any messages.
Command: 73 11 0x 00 Status report: 83 11 0x 00.

10.3.2 Normal

The CAN channel will receive all messages from the network. It will assert the CAN frame ACK bit for all frames it receives without error. Only those frames it receives, where the message ID matches an enabled object ID according to the mask and associated rules, are passed to the Client. Refer to Object ID and Mask for a discussion of object ID and Mask operations.

10.3.3 Transmit Command

The fields and bits construction of a transmit command are shown here. The transmit command is also included in the list of commands at the end of this document.
There are three forms of the transmit command. The number of bytes in the transmit command determines the command formats available for use.
All three formats are acceptable in ascending order.
In other words, a $0x yy … command can be expressed as:
'$0x yy ...'
or as:
'$11 0x yy ...'
or as:
'$12 00 0x yy ...'.
Likewise, an '$11 xx' command can be expressed as '$12 00 xx'.
(But it cannot be expressed using the '0x' form.)

10.3.3.1 Transmit Command Format $0x

The $0x form of the transmit command can be used when the byte count following the header is $0F or less.

10.3.3.2 Transmit Command Format $11 xx

The $11 xx form of the transmit command can be used when the byte count following the header is $FF or less.

10.3.3.3 Transmit Command Format $12 xx yy

The $12 xx yy form of the transmit command can be used when the data byte count of the transmit command is $FFF9 or less.

10.3.3.4 CAN0 and CAN1 Byte Count Limits

The total number of data bytes permitted in a CAN transmit command depends on whether it is a Classical CAN or CAN-FD frame and whether ISO 15765 processing is enabled for the specified transmit object.

10.3.3.5 CAN2 and CAN3 Byte Count Limits

CAN2 and CAN3 support both Classical CAN and CAN-FD formatted frames.
When a transmit command specifies a Classical CAN frame (FDF bit is '0') – then the number of data bytes can be 0 to 8 (inclusive).
When a transmit command specifies a CAN-FD frame (the FDF bit is '1') – then the number of data bytes can be 0 to 64 (inclusive).
In CAN-FD there are fixed data field lengths, listed below. If a transmit command does not contain the proper number of bytes in the data field, the transmit command is in error and the Client will be notified with an error response of the form: '$22 7F xx' and '$32 yy FF'.
There is an optional automatic padding function for CAN2 and CAN3.
This is the '7x 60' Extended Length Padding function.
Do NOT confuse the '7x 60' command with the '7x 27' ISO 15765 padding function.
If Extended Length Padding is enabled, and if the number of data bytes is less than 64, the AVT-425 will automatically pad the data field to the next higher data byte count. For example: You specify a transmit command to CAN2 with 10 data bytes and the pad function is enabled. Then the AVT-425

will add two pad bytes to the data field, raising it to 12 and then queue that CAN frame for transmission.
The '$73 60 0x 0y' command disables and enables the pad function.
The '$73 61 0x yy' command specifies the pad byte value.
CAN-FD data field lengths (data count numbers are decimal):

- 0 to 8 data bytes (variable and inclusive).
- 12 data bytes (fixed).
- 16 data bytes (fixed).
- 20 data bytes (fixed).
- 24 data bytes (fixed).
- 32 data bytes (fixed).
- 48 data bytes (fixed).
- 64 data bytes (fixed).

10.3.4 Receive Response

There are two possible CAN/LIN responses that the AVT-425 can send to the Client.

1. A CAN/LIN message (from another CAN/LIN node).
2. A transmit acknowledgement; aka: a transmit ack.

Both are described in the list of responses at the end of this document.
Regarding messages from the CAN network - there are three possible forms of that receive response.
The number of bytes in the receive response determines the format used by the AVT-425 interface.

10.3.4.1 Receive Response Format: $0x

The $0x form of the receive response is used when the byte count of the response (not including the header byte) is $0F or less.

10.3.4.2 Receive Response Format: $11 xx

The $11 xx form of the receive response is used when the byte count of the response (not including the header byte) is $FF or less.

10.3.4.3 Receive Response Format: $12 xx yy

The $12 xx yy form of the receive response is used when the byte count of the response (not including the header byte) is $FFFF or less.

10.3.4.4 Long Response Only

The Client can use the '5x 06' command to select the receive response format the AVT-425 will use to send received CAN messages to the Client.
When 'long form only' is enabled, the AVT-425 will always use the '12 xx yy' format, regardless of the actual byte count of the response.
This option is missing for ISO15765 receive operations.

10.3.5 Time Stamps

Time stamps for both transmit acknowledgement and received messages can be disabled or enabled using the $5x 08 command.
The time stamp is a four-byte value immediately after the packet header byte but before the CAN channel number.

10.3.5.1 CAN0 and CAN1 Time Stamp Clock

Two time stamp sources are available for each channel.

10.3.5.1.1 Millisecond Time Stamp

A one millisecond time stamp is available. It is a 32-bit free running counter. The time stamp rolls over at $FFFFFFFF.
It is common to the following channels: CAN0, CAN1, LIN0, LIN1, KWP0, KWP1.

10.3.5.1.2 Baud Clock Time Stamp

A time stamp based on the baud clock is available. It is a 16-bit free running counter that is driven by the baud clock for that CAN channel. In other words, the time stamp increment is the inverse of the CAN channel baud rate. For example, if the baud rate is 500 Kbaud, then the time stamp interval is 2 microseconds.
The time stamp clock and counter are separate for CAN0 and CAN1.
The time stamp rolls over at $0000FFFF.

10.3.5.2 CAN2 and CAN3 Time Stamp Clock

A one millisecond time stamp is available. It is a 32-bit free running counter. The time stamp rolls over at $FFFFFFFF.
The time stamp clock for CAN2 and CAN3 is not common to the time stamp clock for the channels listed in CAN0 and CAN1 Time Stamp Clock. However, both time stamp counters are reset at the same time when the AVT-425 application is reset. The Client can use the '5x 05' command to reset all time stamp counters at any time. (Thus achieving some level of synchronization.)

10.3.5.3 Transmit Acknowledgment Description

Refer to Transmit Acknowledgements for a complete description of the transmit ack response to the Client with and without time stamps.

10.4 Object ID and Mask

10.4.1 Configuration – CAN0 and CAN1

Each CAN channel is independent of all other channels.
Each channel has 16 (decimal) message objects numbered: $0 to $F.
Each message object can be configured as either transmit or receive.
Each message object, when configured for receive, can be set for 11 or 29-bit IDs.
Each message object, when configured for receive, has an associated mask.
Each bit of the mask can be set for "must match" or "don't care". The default is all bits are "must match". A '1' in a bit position means "must match".
The combination of the object ID and associated mask give the user flexibility as to what messages are received by the designated object.

10.4.2 Configuration – CAN2 and CAN3

Each CAN channel is independent of all other channels.
Each channel has 16 (decimal) receive message objects numbered: $0 to $F.
Each channel has 16 (decimal) transmit message objects numbered: $0 to $F.
Note the differences between CAN0 / CAN1 and CAN2 / CAN3.

10.4.3 Object ID and Mask Operation

Object IDs and Masks are related.

Object ID0 is associated with Mask0; object ID1 is associated with Mask1, etc.

Using the '$75 2A …' form of the object command specifies an 11-bit ID.
(for objects numbered 0x00 to 0x0F.)
Using the '$77 2A …' form of the object command specifies a 29-bit ID.
(for objects numbered 0x00 to 0x0F.)

A one in a bit position of a mask is a Must Match condition for that bit in the object ID.
A zero in a bit position of a mask is a Don't Care condition for that bit in the object ID.

How the object IDs and masks operate:

A message is received from the network.

- The message ID is passed through the first enabled receive object. Mask and ID are applied.
- If there is a match, the message is passed to the Client.
- If no match, the process is repeated for the next enabled receive object.
- This continues until either a match is made or there are no more enabled receive objects.

10.4.4 Acceptance ID and Mask Notes

The user should configure the Acceptance ID ('7x 2A' command) before setting the Mask ('7x 2C' command). The order does not affect operations, but it will affect the 'look' of the mask response.

10.4.5 CAN0 and CAN1 specifics

For the '7x 18' periodic message setup command, the Client must specify the IDE, RTR, FDF, and BRS bits. They are bits 7, 6, 5, and 4 (respectively) in the channel byte of the '7x 18' command.
For the '7x 2A' receive object setup command the Client can only specify the ID. The Client does not specify the IDE bit as the format of the command indicates 11 or 29-bit ID. The Client cannot specify the RTR, FDF, or BRS bits.
The '7x 2C' receive object mask command does not support the selection of IDE, RTR, FDF, or BRS bits.
A receive object can only be 11-bit or 29-bit.
For all transmit commands the Client must specify the IDE, RTR, FDF, and BRS bits. They are bits 7, 6, 5 and 4 (respectively) of the object byte of the command.

10.4.6 CAN2 and CAN3 specifics

For the '7x 18' periodic message setup command, the Client must specify the IDE, RTR, FDF, and BRS bits. They are bits 7, 6, 5, and 4 (respectively) in the channel byte of the '7x 18' command.
For the '7x 2A' receive object setup command the Client can only specify the ID. The Client does not specify the IDE bit as the format of the command indicates 11 or 29-bit ID. The Client can not specify the RTR, FDF, or BRS bits.
The '7x 2C' receive object mask command does not support the selection of IDE, RTR, FDF, or BRS bits.
A receive object can only be 11-bit or 29-bit.

For all transmit commands the Client must specify the IDE, RTR, FDF, and BRS bits. They are bits 7, 6, 5 and 4 (respectively) of the object byte of the command.

10.4.6.1 ID / Mask Example

Please refer to the following .dwn file packaged with the OMS AVT Diag Tool application:

IDAndMaskExample.dwn

10.5 Setting up CAN0 or CAN1 for operation

The following sequence is recommended for setting a CAN channel for operations.

1. Research the message IDs you want to receive.
2. Ensure the CAN channel is disabled during set-up.
3. Set the CAN channel baud rate.
4. Set up the object(s) you want to use.
5. For each object:
    a. set the mode (receive or transmit),
    b. if receive, set the ID,
    c. if receive, set the mask,
    d. enable the object.
6. There is no need to set-up an object you plan to use for transmit. The transmit command will initialize the object. (Exception: if the object is to be used for transmitting ISO15765 formatted messages, then it must be set-up in advance.)
7. Enable the CAN channel.

10.5.1 Communications Example

Please refer to the following .dwn file packaged with the OMS AVT Diag Tool application:

CommunicationsExample.dwn

Note: The above example requires connecting CAN0 and CAN1 to the same bus.

10.6 Setting up CAN2 or CAN3 for operation

The following sequence is recommended for setting a CAN channel for operations.

1. Research the message IDs you want to receive.
2. Ensure the CAN channel is disabled during set-up.
3. Set the CAN channel baud rates (slow and fast).
4. Set up the object(s) you want to use.
5. For each receive object:
    a. set the ID,
    b. set the mask,
    c. enable the object.
6. Enable the CAN channel.

10.6.1 CAN3 - Classical CAN Communications Example

Please refer to the following .dwn file packaged with the OMS AVT Diag Tool application:

CAN3ClassicalCANCommunicationsExample.dwn

Note: The above example requires connecting CAN2 and CAN3 to the same bus.

10.6.2 CAN-FD Communications Example

Please refer to the following .dwn file packaged with the OMS AVT Diag Tool application:

CANFDCommunicationsExample.dwn

Note: The above example requires connecting CAN2 and CAN3 to the same bus.

10.7 Periodic Message Support

Currently only Type1 periodic messages are supported. Type1 operations is the default mode for all periodic messages.

Type 1 means that each periodic message and its related timer operate together, but independent of all other periodic messages (and their timers).

10.7.1 Periodic Message Number and Numbering

Channels CAN0, CAN1, CAN2, and CAN3 each have 64 (decimal) periodic messages available.
The periodic messages are numbered: $00 to $3F.

10.7.2 Periodic Message Data Field Length

Channels CAN0 and CAN1 are Classical CAN only and are, therefore, limited to data fields of length 0 to 8 byes (inclusive).

For CAN0 and CAN1 – only the '7x 18' command is used to define a periodic message.
Channels CAN2 and CAN3 support CAN-FD periodic messages with data fields of 0 to 64 bytes.

10.7.3 Long Periodic Message Definition and Response

The following only applies to channels CAN2 and CAN3.

The '7x 18' command can be used to define a CAN-FD periodic message with a data field length of 0 to 8 bytes (inclusive).

The '11 bb 2r' or '12 00 bb 2r' long form command is used to define a CAN-FD periodic message with a data field length of 0 to 64 bytes (inclusive).

- A 'long format' command will return a 'long form' response.
- A 'long format' query will return a 'long form' response.
- A '7x 18' command will return an '8x 18' response.
- A '7x 18' query will return an '8x 18' response if the data field is less than or equal to 8 bytes.
- A '7x 18' query will return a 'long form' response if the data field is more than 8 bytes.

Only the following data field lengths are valid for periodic messages. (Lengths are decimal.)

- 0 to 8 data bytes (inclusive).
- 12 data bytes.
- 16 data bytes.

- 20 data bytes.
- 24 data bytes.
- 32 data bytes.
- 48 data bytes.
- 64 data bytes.

10.7.4 Type1 Periodic Messages

Type1 periodic messages operate independently of each other.
When Type1 operations are enabled, each enabled message operates according to its own interval count.

- The message is set up (ID and data field are defined).
- The interval count is defined.

10.7.4.1 Type1 Example

Please refer to the following .dwn file packaged with the OMS AVT Diag Tool application:

CANType1PeriodicMessageExample.dwn

Note: The above example requires connecting CAN2 and CAN3 to the same bus.

10.7.5 Periodic Message Commands

The following commands are used to set up periodic messages:

- 7x 18 Define a periodic message.
- 11 bb 2r / 12 00 bb 2r Define a periodic message (long form, CAN2 and CAN3 only)
- 7x 19 Specify the object for the periodic message. (CAN0 and CAN1 only)
- 7x 1B Periodic message timer increment.
- 7x 1A Periodic message disable/enable.
- 7x 1C Disable all periodic messages.

10.8 Periodic Message Special Functions

There are several special functions available for all CAN periodic messages operating in Type1 mode. These special functions were developed specifically at customer request. Each of the functions are described below.

Each function is available to every CAN periodic message. Each function and each periodic message are independent. In other words, one periodic message can have one function enabled and another periodic message can have another function enabled.
For all of these functions, the data field of a periodic message can be changed 'on the fly'. You do NOT need to disable the message or the function to change anything.

10.8.1 CAN Frame Data Definition

For each periodic message, the CAN frame can contain up to 8 data bytes (or up to 64 for CAN-FD frames).
In the following discussion, Data0 is the first data byte in the CAN frame; or the first data byte onto the network; or the first data byte after the message ID.
Likewise, Data7 (or Data63) is the last data byte of the CAN frame.
Within a byte, the bits are numbered from 0 (least significant bit) to 7 (most significant).

10.8.2 Special Function: Rolling Counter

A rolling counter up to 32 bits wide can be configured with the '7x 40' set of commands. The counter can be set up for any bit location in the message, including across byte boundaries. The counter value is a 32-bit unsigned integer, as is the configurable maximum counter value. The increment/decrement value for the counter is a configurable signed 32-bit value.

10.8.2.1 Rolling Counter Overflow/Underflow

Whenever the periodic message is transmitted, the increment/decrement value is added to the current counter value.

If an overflow occurs, the new counter value is calculated as follows:

$$NewValue = \big((Increment + OldValue) - Maximum\big) - 1$$

Overflow Example:

- Counter Maximum = 255
- Increment = 2
- Current Value = 252

The sequence of values will be: 252, 254, 0, 2

If an underflow occurs, the new counter value is calculated as follows (Decrement is assumed to be negative):

$$NewValue = \big(Maximum - \ (Decrement + OldValue)\big) + 1$$

Underflow Example:

- Counter Maximum = 255
- Decrement = -2
- Current Value = 2

The sequence of values will be: 2, 0, 254, 252

10.8.2.2 Rolling Counter Example

Please refer to the following .dwn files packaged with the OMS AVT Diag Tool application:

RollingCounterSetup_CAN0_Increment_10000.dwn
RollingCounterSetup_CAN1_Decrement_255.dwn

Note: The above examples require connecting CAN0 and CAN1 to the same bus.

10.8.2.3 Rolling Counter Commands

- 7x 40 00 Setup field parameters
- 7x 40 01 Set counter value
- 7x 40 02 Set increment/decrement value
- 7x 40 03 Set maximum value
- 7x 40 04 Enable/Disable Rolling Counter

10.8.3 Special Function: Checksum

The '7x 41' set of commands allows a checksum to be placed within a periodic message. The checksum location and the number of message bytes over which the checksum is calculated are configurable. Currently the following checksum algorithms are supported:

- SAE J1850 CRC
- SAE J1850 CRC with initial and final XOR values set to $00

10.8.3.1 Checksum Example

Please refer to the following .dwn files packaged with the OMS AVT Diag Tool application:

CRCSetup_CAN0_SAEJ1850.dwn

Note: The above example requires connecting CAN0 and CAN1 to the same bus.

CRCSetup_SAEJ1850_LIN0M_LIN1S.dwn

Note: The above example requires connecting LIN0 and LIN1 to the same bus.

10.8.3.2 Checksum Commands

- 7x 41 00 Setup Checksum
- 7x 41 01 Enable/Disable Checksum

10.9 ISO 15765 Support for CAN0 and CAN1

This Section is for channels CAN0 and CAN1 only.
For information about CAN2 and CAN3, refer to ISO 15765 Support for CAN2 and CAN3.

10.9.1 Basic Set-up and Operational Discussion

For one channel, the Client specifies:
A transmit object.
A receive object.
The Client sets-up each object by specifying:
The message ID (11 or 29-bit)
The direction: transmit or receive.
The Client 'pairs' those two objects which enables ISO 15765 processing for those two objects.
The Client specifies if the 'AE' (address extension) is to be used or not.
The Client specifies if message padding is to be used or not.
All CAN frames received through the designated ISO 15765 receive object are processed accordingly.
The PCI byte and pad bytes (if present) are removed.
Only the message ID and valid data are then passed to the Client.
When the Client sends a transmit command it should only include the message ID (11 or 29-bit), the 'AE' byte (if used), and the data bytes. The Client must omit the PCI byte and all pad bytes from the transmit command.
The AVT-425 will process, format, transmit and handle all ISO 15765 handshaking as required.
The Client is not involved in any of it.
Generally speaking, ISO 15765 is used in communications with modules during diagnostic sessions.
Some documentation of module level communications show CAN frames or the data from CAN frames.
This often includes the AE byte (if used), the PCI byte, and the pad bytes.

To transmit those CAN messages, the Client must remove the PCI byte (usually the first byte of the data field) and all pad bytes (if used) what remains forms the basis of a transmit command.
If the user has questions about a specific communications application, please contact Orion Measurement Solutions.
A set-up and operational example is the best way to demonstrate this ISO 15765 capability.
Remember: The AVT-425 will handle all formatting (when transmitting), de-formatting (when receiving) CAN frames, and all communications handshaking (including flow control frames). The Client is not involved in any of the details of ISO 15765 communications.

10.9.2 ISO 15765 for CAN0 and CAN1

Channels CAN0 and CAN1 each have a total of 16 objects (numbered $0 to $F).
An object can be configured for receive or transmit. Only one configuration per object.
Objects are configured for receive using the '7x 2A' command and enabled using the '7x 04 0y 01' command.
Objects are configured for transmit using the '7x 2A' command and enabled using the '7x 04 0y 02' command.

10.9.3 ISO 15765 Initialization and Operation Discussion

Test scenario: The module under test is commanded into diagnostic mode. All communications while in diagnostic mode are ISO 15765 formatted. The module is expecting to:
Receive CAN frames at ID = $246.
Transmit CAN frames at ID = $357.
Address Extension (AE) is NOT being used.
All CAN frames are to be padded with $FF.
The CAN baud rate will be 500 Kbaud.
The user decides to use channel CAN0.
The user decides to use object #2 to transmit and object #3 to receive.

10.9.3.1 ISO 15765 Initialization Example

Please refer to the following .dwn file packaged with the OMS AVT Diag Tool application:

ISO15765InitializationExample.dwn

Note: The above example requires connecting CAN0 and CAN1 to the same bus.

10.9.3.2 Transmit Command Example

The module manufacturer states that to query the module for serial number you send the following CAN frame:
$246 $04 $A1 $A2 $A3 $A4 $FF $FF $FF.
By observation we note that the ID = $246. The PCI byte is $04. The "real" data is "$A1 $A2 $A3 $A4". The last three bytes are pad bytes.
To transmit this message, use the ID and the "real" data to form a transmit command.
The resulting transmit command is:
08 00 02 02 46 A1 A2 A3 A4
That looks much simpler, and is much shorter, than the whole CAN frame that the manufacturer provided.
Quick byte-by-byte explanation:
08: transmit command, upper nibble of '0' means "to the network" and 8 bytes follow.

00: channel CAN0.
02: object #2,
IDE bit = 0 (which means 11-bit ID).
RTR bit = 0 (which means this is a non-RTR CAN frame).
02 46: message ID.
A1 A2 A3 A4: the actual or real data.

Note that the "$04" byte (or PCI byte) is removed.
Note that the pad bytes are removed.
The initialization sequence (above) leaves transmit acknowledgements enabled. Therefore, that transmit command will be followed by the transmit acknowledgement response: 02 00 A2. (which means 02: from the network, 2 bytes follow. 00: channel CAN0. A2: transmit ack, object #2).

10.9.3.3 Receive Response Example

Sending that command (above) to the module should result in the module sending a 14-byte response with the module serial number.
To simulate this, the following transmit command can be sent for CAN1:
11 12 01 02 03 57 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E

But in Classical CAN, a single frame can only hold 8 data bytes. So, the module will use the segmented or multi-frame capability of ISO 15765 to transmit those 14 bytes.
The Client does not need to know this.
The response the Client will receive will be:
11 12 00 03 03 57 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E

Quick byte-by-byte explanation:

11: response packet, next byte is the byte count.
12: $12 bytes to follow.
00: channel CAN0.
03: object #3, IDE and RTR bits are 0.
03 57: received message ID.
01 02 … : the data the module sent.

Note that the Client does not know that three CAN frames were transmitted and other handshaking was conducted between the AVT-425 and the module to obtain this complete response.

10.10 ISO 15765 Support for CAN2 and CAN3

This Section is for channels CAN2 and CAN3 only.
These channels support ISO 15765 operations for both Classical CAN as well as CAN-FD.
For information about CAN0 and CAN1, refer to ISO 15765 support for CAN0 and CAN1.

10.10.1 Basic Set-up and Operational Discussion

For one channel, the Client specifies:

A transmit object.
A receive object.

For channels CAN2 and CAN3, there are 16 receive objects (numbered 0x00 to 0x0F) and 16 transmit objects (numbered 0x00 to 0x0F).
The Client sets-up each object by specifying:

The receive object ID.

The receive object is enabled.
The transmit object ID.

The Client 'pairs' those two objects which enables ISO 15765 processing for those two objects.
The Client specifies if the 'AE' (address extension) is to be used or not.
The Client specifies if message padding is to be used or not.
In the case of an ISO 15675 transaction using CAN-FD – the Client can also specify the maximum number of data bytes in the data field of each frame. This is known as 'max_dlc'. The default is 64 (decimal) bytes of data, maximum, per frame. The '7x 29' command allows the Client to set the maximum number to any of the CAN-FD allowed values.
When CAN-FD is being used, message padding will pad out the frame to the 'max_dlc' limit. (Default value is 64 (decimal) bytes.)
When CAN-FD is being used, if message padding is disabled, the data field is automatically padded to the next higher byte limit (up to the 'max_dlc'). The ISO 15765 pad command sets the value of this pad byte.
All CAN frames received through the designated ISO 15765 receive object are processed accordingly.
The PCI byte and pad bytes (if present) are removed.
Only the message ID and valid data are then passed to the Client.
When the Client sends a transmit command it should only include the message ID (11 or 29-bit), the 'AE' byte (if used), and the data bytes. The Client must omit the PCI byte and all pad bytes from the transmit command.
The AVT-425 will process, format, transmit and handle all ISO 15765 handshaking as required.
The Client is not involved in any of it.
Generally speaking, ISO 15765 is used in communications with modules during diagnostic sessions.
Some documentation of module level communications show CAN frames or the data from CAN frames.
This often includes the AE byte (if used), the PCI byte, and the pad bytes.
To transmit those CAN messages, the Client must remove the PCI byte (usually the first byte of the data field) and all pad bytes (if used) what remains forms the basis of a transmit command.
If the user has questions about a specific communications application, please contact Orion Measurement Solutions.
A set-up and operational example is the best way to demonstrate this ISO 15765 capability.
Examples for both Classical CAN and CAN-FD are provided.
Remember: The AVT-425 will handle all formatting (when transmitting), de-formatting (when receiving) CAN frames, and all communications handshaking (including so-called flow control frames).
The Client is not involved in any of the details of ISO 15765 communications.

10.10.2 ISO 15765 for CAN2 and CAN3

Channels CAN2 and CAN3 each have 16 transmit and 16 receive objects. (Technically, a total of 32 objects.) A receive object can ONLY be configured for receive operations. A transmit object can only be configured for transmit operations.
Objects are configured for receive using the '7x 2A' command and enabled using the '7x 04 yy 01' command.
Objects are configured for transmit using the '7x 17' command. No enable command.

10.10.3 ISO 15765 Initialization and Operation Discussion – Classical CAN

Test scenario: The module under test is commanded into diagnostic mode. All communications while in diagnostic mode are ISO 15765 formatted. The module is expecting to:

Communicate using Classical CAN.
Transmit CAN frames at ID = $246.
Receive CAN frames at ID = $357.

Address Extension (AE) is NOT being used.
All CAN frames are to be padded with $FF.
The CAN baud rate will be 500 Kbaud.

The user decides to:

Use channel CAN2 in Classical CAN mode.
Use object #4 to transmit.
Use object #8 to receive.

10.10.3.1 ISO 15765 Initialization Example – Classical CAN

Please refer to the following .dwn file packaged with the OMS AVT Diag Tool application:

ISO15765InitializationExampleClassicalCAN.dwn

Note: The above example requires connecting CAN2 and CAN3 to the same bus.

10.10.3.2 Transmit Command Example – Classical CAN

The module manufacturer states that to query the module for serial number you send the following CAN frame:
$357 $04 $A1 $A2 $A3 $A4 $FF $FF $FF.
By observation we note that the ID = $246. The PCI byte is $04. The "real" data is "$A1 $A2 $A3 $A4". The last three bytes are pad bytes.
To transmit this message, use the ID and the "real" data to form a transmit command.
The resulting transmit command is:

08 02 04 03 57 A1 A2 A3 A4

That looks much simpler, and is much shorter, than the whole CAN frame that the manufacturer provided.
Quick byte-by-byte explanation:

08: transmit command, upper nibble of '0' means "to the network" and 8 bytes follow.
02: channel CAN2.
04: object #4,
IDE bit = 0 (which means 11-bit ID).
RTR bit = 0 (which means this is a non-RTR CAN frame).
03 57: message ID.
A1 A2 A3 A4: the actual or real data.

Note that the "$04" byte (the PCI byte) is removed.
Note that the pad bytes are removed.
The initialization sequence (above) leaves transmit acknowledgements enabled. Therefore, that transmit command will be followed by the transmit acknowledgement response: 02 02 A4. (which means 02: from the network, 2 bytes follow. 02: channel CAN2. A4: transmit ack, object #4).

10.10.3.3 Receive Response Example – Classical CAN

Sending that command (above) to the module should result in the module sending a 14-byte response with the module serial number.
To simulate this, the following transmit command can be sent for CAN3:
11 12 03 04 02 46 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E

But in Classical CAN, a single frame can only hold 8 data bytes. So, the module will use the segmented or multi-frame capability of ISO 15765 to transmit those 14 bytes.

The Client does not need to know this.
The response the Client will receive will be:
11 12 02 08 02 46 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E

Quick byte-by-byte explanation:

- 11: response packet, next byte is the byte count.
- 12: $12 bytes to follow.
- 02: channel CAN2.
- 08: object #8; IDE, RTR, FDF, and BRS bits are all 0.
- 02 46: received message ID.
- 01 02 … : the data the module sent.

Note that the Client does not know that three CAN frames were transmitted and other handshaking was conducted between the AVT-425 and the module to obtain this complete response.

10.10.4 ISO 15765 Initialization and Operation Discussion –CAN-FD

Test scenario: The module under test is commanded into diagnostic mode. All communications while in diagnostic mode are ISO 15765 formatted. The module is expecting to:

- Communicate using CAN-FD.
- Transmit CAN frames at ID = $246.
- Receive CAN frames at ID = $357.
- Address Extension (AE) is NOT being used.
- All CAN frames are to be padded with $FF.
- The CAN baud rates will be 500 Kbaud and 2 Mbaud.

The user decides to use channel CAN2.
The user decides to use object #4 to transmit and object #8 to receive.

10.10.4.1 ISO 15765 Initialization Example –CAN-FD

Please refer to the following .dwn file packaged with the OMS AVT Diag Tool application:

ISO15765InitializationExampleCANFD.dwn

Note: The above example requires connecting CAN2 and CAN3 to the same bus.

10.10.4.2 Transmit Command Example –CAN-FD

The module manufacturer states that to query the module for serial number you send the following CAN frame:
$357 $00 $14 $01 $02 $03 $04 $05 $06 $07 $08 $09 $0A $0B $0C $0D $0E $0F $10 $11 $12 $13 $14 $FF $FF $FF $FF ……..
(the documentation may include more pad bytes than shown here).
By observation we note that the ID = $357. The PCI byte is $00; this indicates the byte count is the next byte ($14). The '$FF' bytes shown are pad bytes.
To transmit this message, use the ID and the "real" data to form a transmit command.
The resulting transmit command is:

11 18 02 34 03 57 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F 10 11 12 13 14

That looks much simpler, and is much shorter, than the whole CAN frame that the manufacturer provided.
Quick byte-by-byte explanation:

11: transmit command, next byte is command byte count.
18: command byte count.
02: channel CAN2.
34: object #4,
IDE bit = 0 (11-bit ID).
RTR bit = 0 (must be zero).
FDF bit = 1 (CAN-FD).
BRS bit = 1 (use fast baud rate).
03 57: message ID.
01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F 10 11 12 13 14: the actual data.

Note that the "$14" byte (the PCI byte) is removed.
Note that the pad bytes are removed.
The initialization sequence (above) leaves transmit acknowledgements enabled. Therefore, that transmit command will be followed by the transmit acknowledgement response: 02 02 A4. (which means 02: from the network, 2 bytes follow. 02: channel CAN2. A4: transmit ack, object #4).

10.10.4.3 Receive Response Example – CAN-FD

Sending that command (above) to the module should result in the module sending a 14-byte response with the module serial number.

To simulate this, the following transmit command can be sent for CAN3:
11 12 03 34 02 46 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E

The response the Client will receive will be:
11 12 02 38 02 46 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E

Quick byte-by-byte explanation:
11: response packet, next byte is the byte count.
12: $12 bytes to follow.
02: channel CAN2.
38: object #3, IDE = 0; RTR = 0; FDF = 1; BRS = 1.
02 46: received message ID.
01 02 … : the data the module sent.

11 LIN Operations

All of the following LIN discussions apply to all LIN channels, LIN1 and LIN0 (channel numbers 5 and 7, respectively) as well as LIN2-7 (channel numbers $A to $F, inclusive).

11.1 Communications

When enabled, LIN will passively receive all messages from the LIN bus.
The AVT-425 is capable of transmitting to the LIN bus as a Master without data, as a Master with data, or as a Slave with data.

11.2 LIN Message Details

11.2.1 LIN Frame Data Definition

Each LIN frame can contain up to 8 data bytes.
In the following discussions, Data0 is the first data byte in the LIN frame.

Likewise Data7 is the last byte of the LIN frame.
Within a byte, the bits are numbered from 0 (least significant bit) to 7 (most significant).

11.2.2 Message Length

LIN protocol specification revision 2.0 (and later eliminated) defined a relationship between message ID and expected frame length. Current AVT-425 firmware does not provide so-called "ID byte processing" of received messages.

To determine the end of a LIN frame, the AVT-425 watches how much time has elapsed after each byte is received. The operational parameter "receive buffer timeout" ('53 02' command) sets that time interval (in milliseconds). The Client may need to adjust this value for proper reception of LIN bus messages.

Another, related, LIN parameter is the maximum frame time. This timer starts on reception of the sync byte. If this timer expires while message reception is in-progress, the receive buffer is closed and what has been received is sent to the Client.

The maximum frame time is computed based on the LIN bus baud rate and is updated when the baud rate is set.

11.2.3 Checksum

Both Classic and Enhanced checksum methods are supported via the '53 5A' command.
That command selects the checksum that is to be computed and appended to a transmit command.
It is also the checksum method used for checking a received message.
LIN revision 1.3 and earlier use the Classic checksum method.
LIN revision 2.0 and later use the Enhanced checksum.
Also available is "no checksum" option; which means no checksum is computed and no byte is appended to the end of the transmit message.

11.2.4 ID Byte Only Message

If the Master on a LIN bus transmits the ID byte and no module on the bus responds with data, then the message is an ID byte only message. The default state is that the AVT-425 will throw out an ID byte only message and not tell the Client.

The '53 66' command selects whether or not the AVT-425 informs the Client that an ID byte only message was received.

The '53 66 01 01' command causes the AVT-425 to notify the Client that an ID byte only message was received and report the ID byte. The format of the notification is:

23 4A 05 ID

23	error response, 3 bytes follow.
4A	error type: ID only message.
05	channel 5 = LIN.
ID	the received ID byte.

11.2.5 Communications Example

Please refer to the following .dwn file packaged with the OMS AVT Diag Tool program:

LINCommunicationsExample.dwn

Note: The above example requires connecting LIN0 and LIN1 to the same bus.

11.2.6 Time Stamp

Time stamps for both the transmit ack and received messages can be disabled or enabled using the '5x 08' command.
Time stamps are four-bytes (32-bits) taken from a free running 1 msec timer. The time stamp rolls over at $FFFFFFFF.

For both received messages and transmit acks: the time stamp is a four-byte value immediately after the packet header byte; but before the LIN channel number (05).

11.2.6.1 Receive Message Examples

When time stamps are disabled a receive message example is:
08 05 00 25 11 22 33 44
08 header byte, indicates from the network, 8 bytes follow.
05 channel 5 – LIN1
00 status byte indicating no errors detected.
25 message ID.
11 22 33 44 message bytes.
When time stamps are enabled a receive message example is:
0C rr ss tt vv 05 00 25 11 22 33 44
0A header byte, indicates from the network, $A or decimal 10 bytes follow.
rr ss tt vv time stamp.
05 channel 5 – LIN1
00 status byte indicating no errors detected.
25 message ID.
11 22 33 44 message bytes.

11.2.6.2 Transmit Ack Examples

When time stamps are disabled a transmit ack example is:
02 05 40
02 header byte, indicates from the network, 2 bytes follow.
05 channel 5 – LIN1
40 status byte, bit 5 set, indicates from this node.
When time stamps are enabled a transmit ack example is:
06 rr ss tt vv 05 60
04 header byte, indicates from the network, 4 bytes follow.
rr ss tt vv time stamp (xx is the high byte, yy is the low byte).
05 channel 5 – LIN1
40 status byte, bit 5 set, indicates from this node.

11.3 Periodic Message Support

When LIN is enabled, the AVT-425 has the ability to transmit as many as sixteen periodic messages automatically. The operator defines and sets up the desired periodic messages, enables them, and the AVT-425 unit will then transmit those messages, at the defined interval, without any operator intervention.

The AVT-425 will not generate a transmit ack when a periodic message is transmitted, unless the transmit acknowledgement function is enabled ('5x 40' command).

11.3.1 LIN Frame Data Definition

Each LIN frame can contain up to 8 data bytes.
In the following discussions, Data0 is the first data byte in the LIN frame.
Likewise Data7 is the last byte of the LIN frame.
Within a byte, the bits are numbered from 0 (least significant bit) to 7 (most significant).

11.3.2 Modes of Operation

LIN periodic messages are defined as either Master or Slave messages – as specified in the 7x 18 periodic message set-up command.
A periodic message designated as Master will only operate as Type1. Type1 periodic messages are transmitted independent of one another.
A periodic message designated as Slave operate as described in Slave Periodic Message.

11.3.3 Organization of Periodic Messages

There are sixteen (decimal) periodic messages. The periodic messages are numbered: $0 to $F (inclusive).
Each message is independently configured ('7x 18 05' command)
Each message is independently disabled or enabled ('7x 1B 05' command).
Each message has its own time interval ('7x 1A 05' command). The time interval is 1 msec.

11.3.4 Type1 Periodic Message

Type1 periodic messages operate independently of each other.
When Type1 operations are enabled, each enabled message in that group operates according to its own interval count.

The message is set up.
The interval count is defined.
The message is enabled.

A periodic message designated as a Master will be queued for transmission when its timer expires. It will be transmitted as soon as possible after that.

11.3.4.1 Type1 Example

Please refer to the following .dwn file packaged with the OMS AVT Diag Tool program:

LINType1PeriodicMessageExample.dwn

Note: The above example requires connecting LIN0 and LIN1 to the same bus.

11.3.5 Periodic Message Commands

The following commands are used for LIN periodic messages:

- $7x 18 Define a periodic message.
- $7x 1B Periodic message interval.
- $7x 1A Periodic message disable/enable.

- $7x 1C Disable all periodic messages.

11.3.6 Slave Periodic Message

This only applies to periodic messages that are designated as slave when it is set-up.
When a periodic message is set-up and enabled as a slave message ($7x 1A command) it operates independently of a timer. Every time an ID byte is received from the LIN bus, all periodic messages are searched. If a periodic message is a slave and enabled, and if its ID byte matches that just received from the LIN bus, then that message is immediately transmitted into the data field of the LIN frame in progress.

This will happen without Client intervention. The Client will not be informed that the message has been transmitted.

11.4 Periodic Message Special Functions

11.4.1 Special Function: Rolling Counter

The rolling counter function described in Special Function: Rolling Counter is available on LIN channels in addition to CAN channels.

11.4.2 Special Function: Checksum

The checksum function described in Special Function: Checksum is available on LIN channels in addition to CAN channels.

11.5 LIN1 Operations

LIN1 is channel number 5.
LIN1 operation is independent of all other channels.
LIN1 operation is controlled by the $53 69 05 0y command.
LIN1 supports LIN revisions: 1.2, 1.3, 2.0, 2.1, 2.2A.
LIN1 hardware is shared with KWP1 operations. Only one mode can be enabled at a time. Either LIN1 or KWP1, but not both.

11.6 LIN0 operations

LIN0 is channel number 7.
LIN0 operation is independent of all other channels.
LIN0 operation is controlled by the $53 69 07 0z command.
LIN0 supports LIN revisions: 1.2, 1.3, 2.0, 2.1, 2.2A.
LIN0 hardware is shared with KWP0 operations. Only one mode can be enabled at a time. Either LIN0 or KWP0, but not both.

11.7 Commands and Responses

All LIN commands and responses are included in the list at the end of this document.

12 KWP operations

KWP hardware is shared with LIN hardware. Only one mode can be enabled at a time. Either KWP or LIN, but not both. KWP is only available on channel 6 and channel 8 using LIN1 and LIN0 hardware.

12.1 KWP1 operations

KWP1 is channel 6.
KWP1 operation is independent of all other channels.
KWP1 operation is controlled by the $53 69 06 0z command.

12.1.1 KWP1 Operation Limitations

The following KWP operations are supported:

- Receive. Including format byte processing.
- Transmit.
- Periodic messages.
- Fast Initialization.

12.1.2 Commands and Responses

All KWP commands and responses are included in the list at the end of this document.

12.1.3 KWP1 Operation – Quick Intro

A "quick start" guide to KWP operations follows.
Connect the external K-line to pin # 11.
Remember that the AVT-425 and the module must have a common ground and the power supplies must be close in voltage level.
Select KWP1 to be active. Note that this will disable LIN1.
53 69 06 01
Set the K-line baud rate.
53 50 06 0x
At this point any message that shows up on the K-line will be passed to the Client.
Other commands that you may want to use:

- Select checksum type: 5x 4B.
- Transmit inter-byte time: 5x 27.
- Enable/Disable format byte processing: 5x 28.
- Receive buffer time-out (if format byte processing is disabled): 5x 02.

12.1.4 KWP1 Operation – Fast Init

KWP1 operations support 'Fast Init' as defined in ISO14230.
A brief description of how 'Fast Init' operates:

- Fast Init is invoked using the '6x 13 0y rr ss …' command.
  Notes:
    - 'x' is the count of bytes to follow.
    - 'y' is the channel number (6 – for KWP1).
    - 'rr ss …' is the start communications message that the connected module is expecting.
    - A common example is: 65 13 06 81 11 81.
        - '6' – an init command.
        - '5' – five bytes follow.

'13' – fast init.
'6' – channel 6, KWP1.
'81' – physical addressing, one data byte.
'11' – physical address of module.
'81' – start communications service.
the checksum is appended by the AVT-425 (unless that function is disabled).

- Fast init sequence does not proceed until the K-line has been observed to be idle for time 'W5'.
  'W5' default is 300 msec.
  related 'W5' command: 5x 46.
- The AVT-425 will wait for 600 msec for the K-line to be idle for 'W5' before declaring a failure and responding with:
  '2x yy' (transmit watchdog expired)
  and
  '73 13 06 01' (fast init failure).
- The rest of this sequence follows only after the K-line is idle for 'W5'.
- The K-line is pulled low and held low for 25 msec.
  25 msec is the default setting.
  related command: 5x 47.
- The K-line is released to go high for 25 msec.
  25 msec is the default setting.
  related command: 5x 48.
- Transmission of the start communications message begins.
  The communications default baud rate is 10400.
  related command: 5x 50.
- After the start communications message has completed without error, the AVT-425 will respond with ''73 13 06 11' (fast init success).
- During the fast init sequence, most error conditions will be detected. Any error will cause the fast init sequence to terminate and the Client will receive an error response from the AVT-425.

12.2 KWP0 operations

KWP0 is channel 8.
KWP0 operation is independent of all other channels.
KWP0 operation is controlled by the $53 69 08 0z command.

All operations and commands for KWP0 are identical to those for KWP1.
Therefore, refer to KWP1 Operations.

12.3 Commands and Responses

All KWP commands and responses are included in the list at the end of this document.

13 LIN / KWP Bus Supply Voltage

LIN / KWP communications require at least one node to have a passive pull-up resistor between the LIN bus and the supply (usually battery voltage, VBATT). The transceiver uses that same supply to determine the state of the LIN / KWP bus (a 'zero' or 'one' state).

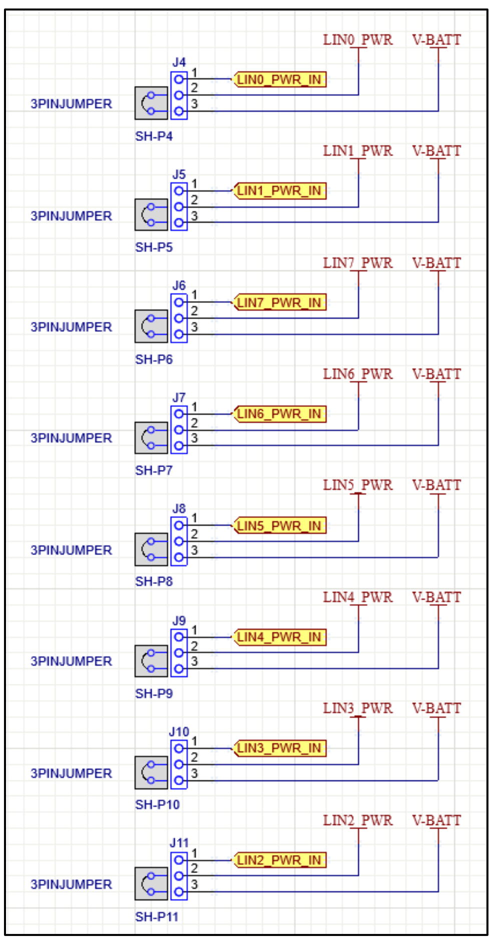
The supply for the AVT-425 LIN / KWP transceivers and that for the connected module is usually the same supply. If they are not the same, then they should, at least, be 'close' in voltage level to one another. Both the AVT-425 and the connected module must share a common ground.

On the AVT-425 board, the LIN / KWP bus pull-up resistor is 1 K ohm [default]. The Client can select a 1K or a 31 K ohm pull-up using the '53 09 0r 0z ' command.

The default configuration for the LIN / KWP bus pull-up supply is the same as the supply to the AVT-425 board. However, the position of the shunt on J4 - J11 allows the user to manually select a separate external LIN / KWP bus supply.

- Shunt across pins 2 to 3 selects LIN / KWP bus supply is the AVT-425 board power
- Shunt across pins 1 to 2 selects LIN / KWP bus supply is pin # 9 of J12 (DB-25P connector).

The LIN / KWP channels bus supply jumper selections are shown in the following circuit diagram:



14 Commands

The first byte of all commands is as follows:

- High nibble, bits b7 - b4, indicates the command category.
- Low nibble, bits b3 – b0 indicates how many bytes are to follow.

The alternate header (1x) commands are the only exceptions to this format.

All transmit command forms are equal in this order (left to right).

0x	=	11 0x	=	12 00 0x
		11 xx	=	12 00 xx
				12 xx yy

14.1 CAN / LIN / KWP Packet for transmission to the network (0x)

CAN0, CAN1

0x 0r qs tt vv ww zz mm nn ... :

- x: count of bytes to follow.
- r: channel: 0, 1
- q:
  - b7: IDE.
    - 0: 11-bit ID.
    - 1: 29-bit ID.
  - b6: RTR.
    - 0: normal frame.
    - 1: RTR true, remote transmit request.
  - b5: 0
  - b4: 0
- s: object number: $0 to $F.
- tt vv: 11-bit ID, right justified.
- tt vv ww zz: 29-bit ID, right justified.
- mm nn ...: data [optional].

CAN2, CAN3

0x 0r qs tt vv ww zz mm nn ... :

- x: count of bytes to follow.
- r: channel: 2, 3.
- q:
  - b7: IDE.
    - 0: 11-bit ID.
    - 1: 29-bit ID.
  - b6: RTR. (only valid for Classical CAN).
    - 0: normal frame.
    - 1: RTR true, remote transmit request.
  - b5: FDF.
    - 0: Classical CAN frame.
    - 1: CAN-FD frame.
  - b4: BRS.
    - 0: data field at normal speed.
    - 1: data field at high speed.
- s: object number: $0 to $F.
- tt vv: 11-bit ID, right justified.
- tt vv ww zz: 29-bit ID, right justified.

mm nn ...: data [optional].

LIN0, LIN1, and LIN2 thru LIN7
0x 0z 0m id aa bb cc … :
x: count of bytes to follow: $0 to $C.
z: channel: 7, 5, and A thru F.
m: slave / master.
0: slave.
1: master.
id: LIN message ID. (Client must include parity bits.)
aa bb cc … data [optional]. Up to 9 bytes to allow the host to specify the checksum if the checksum type is 2 (none)

KWP0, KWP1
0x 0z aa bb cc … :
x: count of bytes to follow.
z: channel: 8, 6.
aa bb cc … data; minimum of one byte.

14.2 CAN packet for transmission to the network; alternate header formats (1x)

CAN0, CAN1
11 xx 0r qs tt vv ww zz mm nn ... :
xx: count of bytes to follow.
r: channel: 0, 1.
q: b7: IDE.
0: 11-bit ID.
1: 29-bit ID.
b6: RTR.
0: normal frame.
1: RTR true, remote transmit request.
b5: 0
b4: 0
s: object number: $0 to $F.
tt vv: 11-bit ID, right justified.
tt vv ww zz: 29-bit ID, right justified.
mm nn ...: data.

CAN0, CAN1
12 xx yy 0r qs tt vv ww zz mm nn ... :
xx yy: count of bytes to follow.
r: channel: 0, 1.
q: b7: IDE.
0: 11-bit ID.
1: 29-bit ID.
b6: RTR.
0: normal frame.
1: RTR true, remote transmit request.
b5: 0
b4: 0

s: object number: $0 to $F.
tt vv: 11-bit ID, right justified.
tt vv ww zz: 29-bit ID, right justified.
mm nn ...: data.

Data byte count limitations CAN0 and CAN1
Maximum is 8 data bytes for non-ISO 15765 operations.
Maximum is $FFF (4095 decimal) for ISO 15765 enabled object.

CAN2, CAN3
11 xx 0r qs tt vv ww zz mm nn ... :
- xx: count of bytes to follow.
- r: channel: 2, 3.
- q:
  - b7: IDE.
    - 0: 11-bit ID.
    - 1: 29-bit ID.
  - b6: RTR. (only valid for Classical CAN frame).
    - 0: normal frame.
    - 1: RTR true, remote transmit request.
  - b5: FDF.
    - 0: Classical CAN frame.
    - 1: CAN-FD frame.
  - b4: BRS.
    - 0: data field at normal speed.
    - 1: data field at high speed.
- s: object number: $0 to $F.
- tt vv: 11-bit ID, right justified.
- tt vv ww zz: 29-bit ID, right justified.
- mm nn ...: data.

CAN2, CAN3
12 xx yy 0r qs tt vv ww zz mm nn ... :
- xx yy: count of bytes to follow.
- r: channel: 2, 3.
- q:
  - b7: IDE.
    - 0: 11-bit ID.
    - 1: 29-bit ID.
  - b6: RTR. (only valid for Classical CAN frame).
    - 0: normal frame.
    - 1: RTR true, remote transmit request.
  - b5: FDF.
    - 0: Classical CAN frame.
    - 1: CAN-FD frame.
  - b4: BRS.
    - 0: data field at normal speed.
    - 1: data field at high speed.
- s: object number: $0 to $F.
- tt vv: 11-bit ID, right justified.
- tt vv ww zz: 29-bit ID, right justified.
- mm nn ...: data.

CAN2, CAN3 Long Periodic Message support
Refer to the '11 bb 2r …' or '12 00 bb 2r …' command for the long periodic message definition command.

Data byte count limitations CAN2 and CAN3
Maximum is 64 bytes with FDF bit = 1 and non-ISO 15765 operations.
Refer to CAN2 and CAN3 Byte Count Limits for the list of valid data field sizes.
Maximum is $2000 (8192 decimal) for ISO 15765 enabled object.

KWP0, KWP1
11 xx 0z aa bb cc … :
xx: count of bytes to follow.
z: channel: 8, 6.
aa bb cc … data. Minimum of one byte. Maximum of 259 bytes.

KWP0, KWP1
12 xx yy 0z aa bb cc … :
xx yy: count of bytes to follow.
z: channel: 8, 6.
aa bb cc … data. Minimum of one byte. Maximum of 259 bytes.

14.3 CAN / LIN Reset (2x)

21 10: Reset CAN0. (Firmware reset.)
21 11: Reset CAN1. (Firmware reset.)
21 12: Reset CAN2. (Firmware reset.)
21 13: Reset CAN3. (Firmware reset.)
21 14: Reset CAN2 and CAN3. (Hardware reset.)
21 20: Reset LIN0. (Firmware reset.)
21 21: Reset LIN1. (Firmware reset.)
21 30: Reset LIN2-7. (Hardware reset.)
21 31: Reset LIN2-7. (Firmware reset.)

14.4 Reserved (3x)

14.5 Reserved (4x)

14.6 CAN / LIN / KWP Configuration (5x)

LIN0, LIN1, KWP0, KWP1, and LIN2 thru LIN7
52 01 0r: Send received checksum to Client, status query.
r: channel: 7, 5, 8, 6, and A thru F.

53 01 0r 0z: Send received checksum to Client.
r: channel: 7, 5, 8, 6, and A thru F.
z: 0: disabled. [Default.]
1: enabled.

LIN0, LIN1, KWP0, KWP1, and LIN2 thru LIN7
52 02 0r: Receive buffer timeout query.

r: channel: 7, 5, 8, 6, and A thru F.

53 02 0r zz: Set receiver buffer timeout.
r: channel: 7, 5, 8, 6, and A thru F.
zz: time in milliseconds.
[Default = 3 msec.]

LIN2 thru LIN7
51 04: Query for status of reset line for LIN2-7 microcontroller.

52 04 00: Hold LIN2-7 microcontroller reset line in the reset state.
52 04 01: Release LIN2-7 microcontroller reset line. Allow it to run.

Not channel specific
51 05: Query for digital output status.

53 05 0r 0s: Control digital output and/or clear time stamp counter.
r: 0: drive output to high impedance.
1: drive output to low impedance.
s: 0: do NOT reset the time stamp counter.
1: do reset the time stamp counter.

CAN2, CAN3
52 06 0r: Query for status of response format for CAN messages received from the CAN bus.
r: channel: 2 or 3.

53 06 0r 0s: Set the response format.
r: channel: 2 or 3.
s: 0: response format depends on message length.
1: always use the long (12 xx yy) format.

CAN0, CAN1, CAN2, CAN3, LIN0, LIN1, KWP0, KWP1, and LIN2 thru LIN7
52 08 0y: Time stamp status query.
y: channel: 0, 1, 2, 3, 7, 5, 8, 6, and A thru F.

53 08 0y 0s: Disable / Enable time stamp.
y: channel: 0, 1, 2, 3, 7, 5, 8, 6, and A thru F.
s: 0: Disable. [Default.]
1: Enable – uses 1 msec timer.
2: Enable – uses native timer.
(CAN0 and CAN1: baud clock.)
(CAN2 and CAN3: not available.)

LIN0, LIN1, KWP0, KWP1, and LIN2 thru LIN7

52 09 0r: Bus pull-up resistor status query.
r: channel: 7, 5, 8, 6, and A thru F.

53 09 0r 0z: Select bus pull-up resistor.
r: channel: 7, 5, 8, 6, and A thru F.
z: 0: 1 K ohm pull-up is enabled.
1: 31 K ohm pull-up is enabled.

LIN0, LIN1
52 1D 0r: Synch break time query.
r: LIN channel: 7, 5.

54 1D 0r yy zz: Set synch break time.
r: LIN channel: 7, 5.
yy zz: increment count. (increment = 1.024 usec).
[Default = $052A = 1322 => 1354 usec.]

KWP0, KWP1
52 27 0r: P4 query. (P4 is transmit inter-byte time.)
r: channel: 8, 6.

54 27 0r zz: Set P4. (P4 is transmit inter-byte time.)
r: channel: 8, 6.
zz: time in milliseconds.
[Default = 5 msec.]

KWP0, KWP1
52 28 0r: Format Byte Processing status query.
r: channel: 8, 6.

54 28 0r 0z: Disable / Enable Format Byte processing.
r: channel: 8, 6.
z: 0: disable.
1: enable.

KWP0, KWP1
52 2A 0r: Query for 'P3' – minimum inter-message time.
r: channel: 8, 6.

53 2A 0r ss: Set 'P3' – minimum inter-message time.
r: channel: 8, 6.
ss: time in milliseconds.
default = 37 = 55 (decimal)

Not channel specific

52 31 0y: TCP parameter query.

54 31 0y 0r 0s: Set TCP parameters.
Note: The parameters are not listed here.
The Client / User should not use this command.

CAN0, CAN1, CAN2, CAN3, LIN0, LIN1, KWP0, KWP1, and LIN2 thru LIN7
52 40 0r Transmit ack status query.
r: channel: 0, 1, 2, 3, 7, 5, 8, 6, and A thru F.

53 40 0r 0y: Disable / Enable transmit acks.
r: channel: 0, 1, 2, 3, 7, 5, 8, 6, and A thru F.
y: 0 disable.
1 enable. [Default.]
2 echo enable.
(option 2 only valid for channels CAN2 and CAN3.)

KWP0, KWP1
52 46 0r: Query for Fast Init 'W5' K-line idle time.
r: channel: 8, 6.

54 46 0r ss tt: Set Fast Init 'W5' K-line idle time.
r: channel: 8, 6.
ss tt: time in milliseconds.
default = 01 2C = 300 (decimal)

KWP0, KWP1
52 47 0r: Query for Fast Init K-line low time.
r: channel: 8, 6.

53 47 0r ss: Set Fast Init K-line low time.
r: channel: 8, 6.
ss: time in milliseconds.
default = 19 = 25 (decimal)

KWP0, KWP1
52 48 0r: Query for Fast Init K-line high time.
r: channel: 8, 6.

53 48 0r ss: Set Fast Init K-line high time.
r: channel: 8, 6.
ss: time in milliseconds.
default = 19 = 25 (decimal)

KWP0, KWP1

52 4B 0r KWP checksum method.
r: channel: 8, 6.

53 4B 0r 0z Select KWP checksum method.
r: channel: 8, 6.
z: 0: no transmit checksum.
1: sum of bytes.
2: sum of bytes, 2's complement.
3: XOR of bytes.

LIN0, LIN1, KWP0, KWP1, and LIN2 thru LIN7
52 50 0r: Baud rate query.
r: channel: 7, 5, 8, 6, and A thru F.

53 50 0r 0z: Set baud rate.
r: channel: 7, 5, 8, 6, and A thru F.
z: 1: 2400 baud.
2: 9600 baud.
3: 19200 baud.
4: 10400 baud.
[Default = 4 => 10400 baud.]

LIN0 and LIN1 only
54 50 0r yy zz: Set baud rate.
r: channel: 7, 5, 8, 6.
yy zz: divisor load.
Baud rate formula. (Numbers shown are decimal.)
Baud rate = 125,000,000 / 32 / yy zz
('yy zz' is converted to decimal)

LIN2 thru LIN7 only
54 50 0r yy zz: Set baud rate.
r: channel: A thru F.
yy zz: divisor load.
Baud rate formula. (Numbers shown are decimal.)
Baud rate = 1,000,000 / yy zz
('yy zz' is converted to decimal)

LIN0, LIN1, and LIN2 thru LIN7
52 52 0r: Maximum frame time query.
r: LIN channel: 7, 5, and A thru F.

53 52 0r zz: Set maximum frame time.
r: LIN channel: 7, 5, and A thru F.
zz: time in milliseconds.
[Default = $13 => 19 msec.]

LIN0, LIN1, and LIN2 thru LIN7
52 5A 0r: Checksum type query.
r: LIN channel: 7, 5, and A thru F.

53 5A 0r 0z: Set checksum type.
r: LIN channel: 7, 5, and A thru F.
z: 0: classic (LIN 1.3).
1: enhanced (LIN 2.0). [Default.]
2: none.

LIN0, LIN1, and LIN2 thru LIN7
52 66 0r: "ID byte only" error response to Client, status query.
r: channel numbers: 7, 5, and A thru F.

53 66 0r 0z: Send "ID byte only" error response to Client.
r: channel numbers: 7, 5, and A thru F.
z: 0: disabled. [Default.]
1: enabled.

KWP0, KWP1
52 66 0r: "One byte only" error response to Client, status query.
r: channel numbers: 8, 6.

53 66 0r 0z: Send "One byte only" error response to Client.
r: channel numbers: 8, 6.
z: 0: disabled. [Default.]
1: enabled.

LIN0, LIN1, KWP0, KWP1, and LIN2 thru LIN7
52 69 0r: Secondary operations status query.
r: channel numbers: 7, 5, 8, 6, and A thru F.

53 69 0r 0z: Set secondary operations.
r: channel numbers: 7, 5, 8, 6, and A thru F.
z: 0: disabled. [Default.]
1: enabled.

Notes:
LIN0 and KWP0 operations are mutually exclusive – they share hardware.
LIN1 and KWP1 operations are mutually exclusive – they share hardware.

Not channel specific
51 6A: Query for Netburner CPU heart beat LED blink rate.

53 6A yy zz: Set Netburner CPU heart beat LED blink rate.
yy zz: LED half period time, msec.
[Default = $01F4 => 500 msec.]

Not channel specific
51 6B: Query for LIN2-7 CPU heart beat LED blink rate.

53 6B yy zz: Set LIN2-7 CPU heart beat LED blink rate.
yy zz: LED half period time, msec.
[Default = $01F4 => 500 msec.]

Not channel specific
51 6C: Query for CAN2/3 CPU heart beat LED blink rate.

53 6C yy zz: Set CAN2/3 CPU heart beat LED blink rate.
yy zz: LED half period time, msec.
[Default = $01F4 => 500 msec.]

LIN0, LIN1, KWP0, KWP1, and LIN2 thru LIN7
52 7E 0r: Short to ground counter reset value query.
r: channel numbers: 7, 5, 8, 6, and A thru F.

54 7E 0r yy zz: Short to ground counter reset value.
r: channel numbers: 7, 5, 8, 6, and A thru F.
yy zz: counter reset value.
[Default = $0100 => 256.]

Not channel specific
51 80: Query for stored (non-volatile) start-up parameters.
55 80 rr ss tt vv: Set stored (non-volatile) start-up parameters.
rr ss tt vv is the following bit map.
b31: '1' reserved, not used.
b30: '1' reserved, not used.
b29: '1' reserved, not used.
b28: '1' reserved, not used.
b27: '1' reserved, not used.
b26: '1' reserved, not used.
b25: '1' reserved, not used.
b24: '1' reserved, not used.
b23: '1' reserved, not used.
b22: '1' reserved, not used.
b21: '1' reserved, not used.
b20: '1' reserved, not used.
b19: '1' reserved, not used.
b18: '1' reserved, not used.
b17: '1' reserved, not used.
b16: '1' reserved, not used.
b15: '1' reserved, not used.
b14: '1' reserved, not used.

	b13:	'1' reserved, not used.	
	b12:	'1' reserved, not used.	
	b11:	'1' reserved, not used.	
	b10:	'1' reserved, not used.	
	b09:	'1' reserved, not used.	
	b08:	'1' reserved, not used.	
	b07:	CAN3 transceiver control	1 = transceiver enabled. 0 = transceiver disabled.
	b06:	CAN2 transceiver control	1 = transceiver enabled. 0 = transceiver disabled.
	b05:	CAN1 transceiver control	1 = transceiver enabled. 0 = transceiver disabled.
	b04:	CAN0 transceiver control	1 = transceiver enabled. 0 = transceiver disabled.
	b03:	CAN3 termination	1 = termination enabled. 0 = termination disabled.
	b02:	CAN2 termination	1 = termination enabled. 0 = termination disabled.
	b01:	CAN1 termination	1 = termination enabled. 0 = termination disabled.
	b00:	CAN0 termination	1 = termination enabled. 0 = termination disabled.
52 81 01:	Query Serial Number		

14.7 KWP Initialization (6x)

KWP0, KWP1

6x 13 0r ss tt vv … conduct a 'Fast Init' (ISO 14230)

r: channel numbers: 8, 6.

ss tt vv … : the fast init message (usually the start comms message).

14.8 CAN / LIN / KWP configuration (7x)

CAN2, CAN3

72 01 0r: Request ISO15765 buffer time out reset value.

r: channel: 2, 3.

74 01 0r xx yy: Set the ISO15765 buffer time out reset value.

r: channel: 2, 3.

xx yy: reset value (msec).

The Client / User should NOT modify this value
The command should only be used for testing and debugging.

CAN2, CAN3

71 03: Request number of 1 usec wait intervals.

73 03 xx yy: Set the number of 1 usec wait intervals.

xx yy: count of wait intervals.

The Client / User should NOT modify this value
The command should only be used for testing and debugging.

CAN0, CAN1
72 04 0r: Object status query.
r: channel: 0, 1.

74 04 0r 0y 0z Disable / Enable object.
r: channel: 0, 1.
y: object number $0 to $F.
z: 0: object disabled. [Default.]
1: object enabled for receive.
2: object enabled for transmit.

CAN2, CAN3
72 04 0r: Object status query.
r: channel: 2, 3.

74 04 0r yy 0z Disable / Enable object.
r: channel: 2, 3.
yy: object number: $0 to $F.
z: 0: object disabled. [Default.]
1: object enabled for receive.

CAN0, CAN1
73 07 0r 0z: Query for object transmit status.
r: channel: 0, 1.
z: object number: $0 to $F.

74 07 0r 0z 0w: Set object transmit status.
r: channel: 0, 1.
z: object number: $0 to $F.
w: 0: abort transmission.
1: transmit the object.

CAN2, CAN3
71 09: Request received frame processing limit.

72 09 yy: Set the number of received frames processed before exiting the receive manager.
yy: count.

The Client / User should NOT modify this value
The command should only be used for testing and debugging.

CAN0, CAN1, CAN2, CAN3
Channels 0 and 1: The response is always of the form '83 0A'.
Channels 2 and 3: The response is always of the form '84 0A'
regardless of the form of the command.

72 0A 0r: Baud rate query.
- r: channel: 0, 1, 2, 3.

73 0A 0r yy: Set baud rate.
- r: channel: 0, 1, 2, 3.
- yy:
  - 00: Client specified using 74 0B 0x rr ss command.
  - 01: 1 Mbps.
  - 02: 500 Kbps. [Default.]
  - 03: 250 Kbps.
  - 04: 125 Kbps.
  - 0A: 33.333 Kbps.
  - 0B: 83.333 Kbps.

74 0A 0r yy zz: Set baud rate.
- r: channel: 2, 3.
- yy:
  - 00: Client specified using 74 0B 0x rr ss command.
  - 01: 1 Mbps.
  - 02: 500 Kbps. [Default.]
  - 03: 250 Kbps.
  - 04: 125 Kbps.
  - 0A: 33.333 Kbps.
  - 0B: 83.333 Kbps.
- zz:
  - 00: Client specified using 74 0B 0x rr ss command.
  - 01: 1 Mbps.
  - 02: 500 Kbps. [Default.]
  - 03: 250 Kbps.
  - 04: 125 Kbps.
  - 0A: 33.333 Kbps.
  - 0B: 83.333 Kbps.
  - 0C: 2 Mbps.
  - 0D: 4 Mbps.
  - 0E: 5 Mbps.
  - 0F: 8 Mbps.

CAN0, CAN1
72 0B 0r: Query for Bit Timing Register. (CANCTRL).
- r: channel: 0, 1.

76 0B 0r ss tt vv ww: Set Bit Timing Registers (CANCTRL).
- r: channel: 0, 1.
- ss tt vv ww: Bit timing register.

Only bits 31:16 and 2:0 are used. All other bits are masked off (zero) before being written to the register. All bits are reported during a query.

Refer to Motorola / Freescale / NXP MCF5441X (CPU) FlexCAN chapter for detailed information about this register (CANCTRL).

Bit definitions:

31:24	Prescaler division factor. Actual divisor is the value written plus one.
23:22	Resynchronization jump width.
21:19	Phase buffer segment 1.
18:16	Phase buffer segment 2.
2:0	Propagation segment.

Contact Orion Measurement Solutions if you have questions and need to use this command.

CAN2, CAN3

72 0B 0r: Query for Slow and Fast bit timing registers (NBTP and DBTP).
- r: channel: 2, 3.

7A 0B 0r s3 s2 s1 s0 f3 f2 f1 f0: Set Slow and Fast bit timing registers (NBTP and DBTP).
- r: channel: 2, 3.
- s3 s2 s1 s0: bits 31:00 of NBTP
- f3 f2 f1 f0: bits 31:00 of DBTP

Refer to Bosch MCAN manual for register definitions.
All bits are written and read.

Contact Orion Measurement Solutions if you have questions and need to use this command.

CAN0, CAN1, CAN2, CAN3

72 0E 0r: Query for ISO 15765 outbound flow control separation time (ms).
- r: channel: 0, 1, 2, 3.

73 0E 0r zz: Set ISO 15765 outbound flow control separation time.
- r: channel: 0, 1, 2, 3.
- zz: separation time to use in an outbound flow control frame. time in milliseconds. Valid range: $00 to $7F. [Default = 0.]

CAN0, CAN1, CAN2, CAN3

72 11 0r: Query for channel operation status.
- r: channel: 0, 1, 2, 3.

73 11 0r 0z: Set channel operation state.
- r: channel: 0, 1, 2, 3.
- z:
  - 0: channel disabled. [Default]
  - 1: channel enabled for normal operations.

CAN1
71 12: Query for Single Wire CAN (SWC) transceiver status.

72 12 0y: Set SWC transceiver mode.
y: 0: Sleep mode.
1: High speed mode.
2: Wake up mode.
3: Normal mode. [Default.]

CAN0, CAN1, CAN2, CAN3
72 13 0r: Query for operational status of CAN transceiver, 2-wire.
r: channel: 0, 1, 2, 3.

73 13 0r 0y: Set operational status of CAN transceiver, 2-wire.
r: channel: 0, 1, 2, 3.
y: 0 – transceiver disabled.
1 – transceiver enabled. [Default]

CAN2, CAN3 (Transmit object only.)
73 17 0r zz Query for transmit object configuration.
r: CAN channel: 2, 3.
zz: object number: $0 to $F.

75 17 0r wz tt vv Set transmit object configuration.
r: CAN channel: 2, 3.
w: b7: 0
b6: 0
b5: 0 = Classical CAN frame (FDF = 0)
1 = CAN-FD frame (FDF = 1)
b4: 0 = data at normal speed (BRS = 0)
1 = data at high speed (BRS = 1)
z: object number: $0 to $F.
tt vv: 11-bit ID, right justified.

77 17 0r wz tt vv mm nn Set transmit object configuration.
r: CAN channel: 2, 3.
w: b7: 0
b6: 0
b5: 0 = Classical CAN frame (FDF = 0)
1 = CAN-FD frame (FDF = 1)
b4: 0 = data at normal speed (BRS = 0)
1 = data at high speed (BRS = 1)
z: object number: $0 to $F.
tt vv mm nn: 29-bit ID, right justified.

CAN0, CAN1

73 18 0r pp: Periodic message set-up query.
- r: channel: 0, 1.
- pp: message number: $00 to $3F.

7x 18 yr pp tt vv ww zz mm nn ... Periodic message set-up.
- y:
  - b7: IDE.
    - 0: 11-bit ID.
    - 1: 29-bit ID.
  - b6: RTR.
    - 0: normal frame.
    - 1: RTR true, remote transmit request.
  - b5: 0
  - b4: 0
- r: channel: 0, 1.
- pp: message number: $00 to $3F.
- tt vv: 11-bit ID, right justified.
- tt vv ww zz: 29-bit ID, right justified.
- mm nn ...: data field (optional; 0 to 8 bytes).

CAN2, CAN3

73 18 0r pp: Periodic message set-up query.
- r: channel: 2, 3.
- pp: message number: $00 to $3F.

7x 18 yr pp tt vv ww zz mm nn ... Periodic message set-up.
- y:
  - b7: IDE.
    - 0: 11-bit ID.
    - 1: 29-bit ID.
  - b6: RTR. (only valid for Classical CAN frame).
    - 0: normal frame.
    - 1: RTR true, remote transmit request.
  - b5: FDF.
    - 0: Classical CAN frame.
    - 1: CAN-FD frame.
  - b4: BRS.
    - 0: data field at normal speed.
    - 1: data field at high speed.
- r: channel: 2, 3.
- pp: message number: $00 to $3F.
- tt vv: 11-bit ID, right justified.
- tt vv ww zz: 29-bit ID, right justified.
- mm nn ...: data field (optional; 0 to 8 bytes).

CAN2, CAN3

Long form command for data field byte count of 0 to 64 (decimal) inclusive.

Both forms are valid.

Long Form Query

11 03 2r 00 pp

12 00 03 2r 00 pp

11:            long header.
12 00:   long header.
03:            three bytes follow.
2:             long periodic message query.
r: channel number: 2, 3.
00
pp: message number: $00 to $3F.

Long Form Command
11 bb 2r y0 pp tt vv ww zz mm nn ...
12 00 bb 2r y0 pp tt vv ww zz mm nn ...
11:            long header.
12 00:   long header.
bb: count of bytes to follow.
2:             long periodic message command.
r: channel number: 2, 3.
y: b7:   IDE   0:   11-bit.
                           1:   29-bit.
               b6:   RTR   0:   is NOT an RTR frame.
                           1:   is an RTR frame.
               b5:   FDF   0:   Classical CAN.
                           1:   CAN-FD.
               b5:   BRS   0:   normal speed data field.
                           1:   high speed data field.
               b4:   0
pp: message number: $00 to $3F.
tt vv: 11-bit id.
tt vv ww zz: 29-bit id.
mm nn ... :   data field (0 to 64 bytes, inclusive).
               Refer to Long Periodic Message Definition and Response for valid data field lengths.

LIN0, LIN1, and LIN2 thru LIN7
73 18 0r pp: Periodic message set-up query.
               r: channel: 7, 5, and A thru F.
               pp: message number: $0 to $F.

7x 18 0r pp 0m rr ss tt …   Periodic message set-up.
               r: channel: 7, 5, and A thru F.
               pp: message number: $0 to $F.
               m: 0:   slave.
                              1:   master.
               rr ss tt ...: data field. [optional]

KWP0, KWP1
73 18 0x 0p: Periodic message set-up query.
               x: channel number: 8, 6.
               p: message number: $0 to $F.

7x 18 0x 0p rr ss tt …   Periodic message set-up.
               x: channel number: 8, 6.

p: message number: $0 to $F.
rr ss tt ...: data field.

CAN0, CAN1
73 19 0r pp: Query for object assignment for CAN periodic message.
r: channel: 0, 1.
pp: message number: $00 to $3F.

74 19 0r pp 0y: Assign object to CAN periodic message.
r: channel: 0, 1.
pp: message number: $00 to $3F.
y: object number: $0 to $F.

CAN0, CAN1, CAN2, CAN3
73 1A 0r zz: Periodic message disable/enable status query.
r: channel: 0, 1, 2, 3.
zz: message number: $00 to $3F.

74 1A 0r zz 0v: Periodic message disable/enable.
r: channel: 0, 1, 2, 3.
zz: message number: $00 to $3F.
v: 0 disabled. [Default.]
1 enabled.

LIN0, LIN1, KWP0, KWP1, and LIN2 thru LIN7
73 1A 0r zz: Periodic message disable/enable status query.
r: channel: 7, 5, 8, 6, and A thru F.
zz: message number: $0 to $F.

74 1A 0r zz 0v: Periodic message disable/enable.
r: channel: 7, 5, 8, 6, and A thru F.
zz: message number: $0 to $F.
v: 0 disabled. [Default.]
1 enabled.

CAN0, CAN1, CAN2, CAN3
73 1B 0r zz: Periodic message interval count status query.
r: channel: 0, 1, 2, 3.
zz: message number: $00 to $3F.

75 1B 0r zz vv ww: Periodic message interval count.
r: channel: 0, 1, 2, 3.
zz: message number: $00 to $3F.
ww vv: time in milliseconds.
[Default = $03E8 => 1000 msec.]

LIN0, LIN1, KWP0, KWP1, and LIN2 thru LIN7

73 1B 0r zz: Periodic message interval count status query.
r: channel: 7, 5, 8, 6, and A thru F.
zz: message number: $0 to $F.

75 1B 0r zz vv ww: Periodic message interval count.
r: channel: 7, 5, 8, 6, and A thru F.
zz: message number: $0 to $F.
ww vv: time in milliseconds.
[Default = $03E8 => 1000 msec.]

CAN0, CAN1, CAN2, CAN3, LIN0, LIN1, KWP0, KWP1, and LIN2 thru LIN7
72 1C rr: Disable all periodic message for channel 'r'.
0r: channel: 0, 1, 2, 3, 7, 5, 8, 6, and A thru F.
rr: 'FF' – disable all messages, all channels.

CAN2, CAN3
72 1F 0r: Query for receive frame processing status.

73 1F 0r 00: Do not process non-ISO15765 received frames.
73 1F 0r 01: Do process non-ISO15765 received frames.

CAN0, CAN1, CAN2, CAN3
72 25 0r: Query for flow control additional separation time.
r: channel: 0, 1, 2, 3.

73 25 0r ss: Set the flow control additional separation time.
r: channel: 0, 1, 2, 3.
ss: additional time, in milliseconds,
to add to the responding node separation time.

CAN0, CAN1
73 27 0r 0z: Query for ISO 15765 padding status.
r: channel: 0, 1, 2, 3.
z: object number: $0 to $F.

74 27 0r 0z 0v: Disable / enable ISO 15765 padding status.
r: channel: 0, 1, 2, 3.
z: object number: $0 to $F.
v: 0: disable.
1: enable. [Default.]

75 27 0r 0z 0v ww: Disable / enable ISO 15765 padding status.
r: channel: 0, 1, 2, 3.
z: object number: $0 to $F.
v: 0: disable.
1: enable. [Default.]

ww: pad byte.
[Default = $FF.]

CAN2, CAN3

73 27 0r zz: Query for ISO 15765 padding status
r: channel: 0, 1, 2, 3.
zz: object number: 0x0 to 0xF.

74 27 0r zz 0v: Disable / enable ISO 15765 padding status.
r: channel: 0, 1, 2, 3.
zz: object number: 0x0 to 0xF.
v: 0: disable.
1: enable. [Default.]

75 27 0r zz 0v ww: Disable / enable ISO 15765 padding status.
r: channel: 0, 1, 2, 3.
zz: object number: 0x0 to 0xF.
v: 0: disable.
1: enable. [Default.]
ww: pad byte.
[Default = $FF.]

CAN0, CAN1

72 28 0r: Query for pairing status, all objects.
r: channel: 0, 1, 2, 3.

73 28 0r 0y: Disable object pairing, object 'y' and its mate.
Disable ISO 15765 operations.
r: channel: 0, 1, 2, 3.
y: object number: $0 to $F.

74 28 0r 0y 0s: Pair the two objects for ISO 15765 operations.
r: channel: 0, 1, 2, 3.
y: object number: $0 to $F.
s: object number: $0 to $F.

75 28 0r 0y 0s ww: Pair the two objects for ISO 15765 operations and set 'AE' byte.
r: channel: 0, 1, 2, 3.
y: object number: $0 to $F.
s: object number: $0 to $F.
ww: set 'AE' byte.

CAN2, CAN3

72 28 0r: Query for pairing status, all objects.
r: channel: 0, 1, 2, 3.

73 28 0r 0y: Disable object pairing, object 'y' and its mate.
Disable ISO 15765 operations.
r: channel: 0, 1, 2, 3.

y: object number: 0x0 to 0xF.

74 28 0r 0y 0s: Pair the two objects for ISO 15765 operations.
r: channel: 0, 1, 2, 3.
y: object number: 0x0 to 0xF.
ss: object number: 0x0 to 0xF.

75 28 0r 0y 0s ww: Pair the two objects for ISO 15765 operations and set 'AE' byte.
r: channel: 0, 1, 2, 3.
y: object number: 0x0 to 0xF.
s: object number: 0x0 to 0xF.
ww: set 'AE' byte.

CAN2, CAN3
73 29 0r 0y: Query for 'max_dlc'.
r: channel: 2, 3.
y: object number: 0x0 to 0xF.

74 29 0r 0y ss: Set 'max_dlc'. (Only used with ISO15765 processing.)
r: channel: 2, 3.
y: object number: 0x0 to 0xF.
ss: only the following values are valid (hex digits):
08, 0C, 10, 14, 18, 20, 30, 40.

CAN0, CAN1
73 2A 0r 0z: Report object configuration.
r: channel: 0, 1.
z: object number: $0 to $F.

75 2A 0r yz ss tt: Configure object for 11-bit ID.
r: channel: 0, 1.
y: b7: 0.
b6: RTR.
0: normal frame.
1: RTR true, remote transmit request.
b5: 0
b4: 0
z: object number: $0 to $F.
ss tt: 11-bit ID.

77 2A 0r yz ss tt vv ww: Configure object for 29-bit ID.
r: channel: 0, 1.
y: b7: 0.
b6: RTR.
0: normal frame.
1: RTR true, remote transmit request.
b5: 0
b4: 0

z: object number: $0 to $F.
ss tt vv ww: 29-bit ID.

CAN2, CAN3
73 2A 0r 0z: Report object configuration.
r: channel: 2, 3.
z: object number: $0 to $F.

75 2A 0r 0z ss tt: Set object 11-bit ID.
r: channel: 2, 3.
z: object number: $0 to $F.
ss tt: 11-bit ID, right justified.

77 2A 0r 0z ss tt vv ww: Set object 29-bit ID.
r: channel: 2, 3.
z: object number: $0 to $F.
ss tt vv ww: 29-bit ID, right justified.

CAN0, CAN1
73 2C 0r 0z: Report mask.
r: channel: 0, 1.
z: mask number: $0 to $F.

1: Bit must match.
0: Bit is don't care.

75 2C 0r 0z ss tt: Specify 11-bit mask.
r: channel: 0, 1.
z: mask number: $0 to $F.
ss tt: mask value, 11-bit.
[Default = $7FF.]

77 2C 0r 0z ss tt vv ww: Specify 29-bit mask.
r: channel: 0, 1.
z: mask number: $0 to $F.
ss tt vv ww: Mask value, 29-bit.
[Default = $1FFFFF.]

CAN2, CAN3
73 2C 0r 0z: Report mask.
r: channel: 2, 3.
z: mask number: $0 to $F.

1: Bit must match.
0: Bit is don't care.

75 2C 0r 0z ss tt: Specify 11-bit mask.
r: channel: 2, 3.
z: mask number: $0 to $F.

ss tt: mask value, 11-bit.
[Default = $7FF.]

77 2C 0r 0z ss tt vv ww: Specify 29-bit mask.
r: channel: 2, 3.
z: mask number: $0 to $F.
ss tt vv ww: mask value, 29-bit.
[Default = $1FFFFF.]

CAN0, CAN1, CAN2, CAN3
73 30 0r 0z: Query for ISO 15765 'AE' status and byte.
r: channel: 0, 1, 2, 3.
z: object number: $0 to $F.

74 30 0r 0z 0s: Disable / Enable ISO 15765 'AE' operation.
r: channel: 0, 1, 2, 3.
z: object number: $0 to $F.
s: 0 = disable, 1 = enable

75 30 0r 0z 0s ww: Disable / Enable ISO 15765 'AE' operation; specify 'AE' byte.
r: channel: 0, 1, 2, 3.
z: object number: $0 to $F.
s: 0 = disable, 1 = enable.
ww: 'ae' byte.

CAN0, CAN1, CAN2, CAN3
74 40 0p 0r zz: Query for Rolling Counter parameter for Periodic Message
p: parameter.
0 = field
1 = value
2 = increment/decrement
3 = maximum
4 = enable/disable
r: channel: 0, 1, 2, 3, 5, 7, $A thru $F.
zz: message number: $00 to $3F.

77 40 00 0r zz ww BB 0b: Setup Rolling Counter Field Parameters for Periodic Message
r: channel: 0, 1, 2, 3, 5, 7, $A thru $F.
zz: message number: $00 to $3F.
ww: rolling counter width: $00 to $20
BB: byte number of the counter lsb: $00 to $3F
b: bit number of the counter lsb: $0 to $7

78 40 01 0r zz VV VV VV VV: Set Rolling Counter Value for Periodic Message
r: channel: 0, 1, 2, 3, 5, 7, $A thru $F.
zz: message number: $00 to $3F.
VV VV VV VV: 32-bit unsigned counter value

78 40 02 0r zz II II II II: Set Rolling Counter Increment/Decrement Value for Periodic Message

r: channel: 0, 1, 2, 3, 5, 7, $A thru $F.
zz: message number: $00 to $3F.
II II II II: 32-bit signed increment/decrement value

78 40 03 0r zz MM MM MM MM: Set Rolling Counter Maximum Value for Periodic Message
r: channel: 0, 1, 2, 3, 5, 7, $A thru $F.
zz: message number: $00 to $3F.
MM MM MM MM: 32-bit unsigned maximum value

75 40 04 0r zz 0e: Enable/Disable Rolling Counter for Periodic Message
r: channel: 0, 1, 2, 3, 5, 7, $A thru $F.
zz: message number: $00 to $3F.
e: enable. 0 = disable, 1 = enable.

LIN0 thru LIN7
74 40 0p 0r zz: Query for Rolling Counter parameter for Periodic Message
p: parameter.
0 = field
1 = value
2 = increment/decrement
3 = maximum
4 = enable/disable
r: channel: 0, 1, 2, 3, 5, 7, $A thru $F.
zz: message number: $0 to $F.

77 40 00 0r zz ww BB 0b: Setup Rolling Counter Field Parameters for Periodic Message
r: channel: 0, 1, 2, 3, 5, 7, $A thru $F.
zz: message number: $00 to $3F.
ww: rolling counter width: $00 to $20
BB: byte number of the counter lsb: $0 to $F
b: bit number of the counter lsb: $0 to $7

78 40 01 0r zz VV VV VV VV: Set Rolling Counter Value for Periodic Message
r: channel: 0, 1, 2, 3, 5, 7, $A thru $F.
zz: message number: $0 to $F.
VV VV VV VV: 32-bit unsigned counter value

78 40 02 0r zz II II II II: Set Rolling Counter Increment/Decrement Value for Periodic Message
r: channel: 0, 1, 2, 3, 5, 7, $A thru $F.
zz: message number: $0 to $F.
II II II II: 32-bit signed increment/decrement value

78 40 03 0r zz MM MM MM MM: Set Rolling Counter Maximum Value for Periodic Message
r: channel: 0, 1, 2, 3, 5, 7, $A thru $F.
zz: message number: $0 to $F.
MM MM MM MM: 32-bit unsigned maximum value

75 40 04 0r zz 0e: Enable/Disable Rolling Counter for Periodic Message
r: channel: 0, 1, 2, 3, 5, 7, $A thru $F.
zz: message number: $0 to $F.

e: enable. 0 = disable, 1 = enable.

CAN0, CAN1, CAN2, CAN3

74 41 0p 0r zz: Query for Checksum parameter for Periodic Message
- p: parameter.
  - 0 = settings
  - 1 = enable/disable

r: channel: 0, 1, 2, 3, 5, 7, $A thru $F.
- zz: message number: $00 to $3F.

77 41 00 0r zz TT BB CC: Setup Checksum Settings Parameters for Periodic Message
- r: channel: 0, 1, 2, 3, 5, 7, $A thru $F.
- zz: message number: $00 to $3F.
- TT: checksum type:
  - 0 = SAE_J1850
  - 1 = SAE_J1850_ZERO
- BB: byte number of the checksum lsb: $00 to $3F
- CC: message byte count for the checksum: $00 to $3F

75 41 01 0r zz 0e: Enable/Disable Checksum for Periodic Message
- r: channel: 0, 1, 2, 3, 5, 7, $A thru $F.
- zz: message number: $00 to $3F.
- e: enable. 0 = disable, 1 = enable.

LIN0 thru LIN7

74 41 0p 0r zz: Query for Checksum parameter for Periodic Message
- p: parameter.
  - 0 = settings
  - 1 = enable/disable
- r: channel: 0, 1, 2, 3, 5, 7, $A thru $F.
- zz: message number: $0 to $F.

77 41 00 0r zz TT BB CC: Setup Checksum Settings Parameters for Periodic Message
- r: channel: 0, 1, 2, 3, 5, 7, $A thru $F.
- zz: message number: $0 to $F.
- TT: checksum type:
  - 0 = SAE_J1850
  - 1 = SAE_J1850_ZERO
- BB: byte number of the checksum lsb: $00 to $3F
- CC: message byte count for the checksum: $00 to $3F

75 41 01 0r zz 0e: Enable/Disable Checksum for Periodic Message
- r: channel: 0, 1, 2, 3, 5, 7, $A thru $F.
- zz: message number: $0 to $F.
- e: enable. 0 = disable, 1 = enable.

CAN1

71 45: Query for CAN1 transceiver.

72 45 01: Set CAN1 transceiver to single wire CAN (SWC).
72 45 02: Set CAN1 transceiver to 2-wire CAN. [Default.]

CAN2, CAN3
72 60 0r Query for status of extended data length padding.
r: channel: 2, 3.

73 60 0r 0y: Set status of extended data length padding.
r: channel: 2, 3.
y: 0: disabled. [Default.]
1: enabled.

CAN2, CAN3
72 61 0r Query for pad byte value.
r: channel: 2, 3.

73 61 0r yy: Set pad byte values.
r: channel: 2, 3.
yy: pad byte value. [Default = $EE.]

CAN0, CAN1, CAN2, CAN3
72 62 0r: Query for CAN 2-wire bus termination status.
r: channel: 0, 1, 2, 3.

73 62 0r 0y: Set CAN 2-wire bus termination.
r: channel: 0, 1, 2, 3.
y: 0: disabled.
1 enabled. [Default.]

14.9 Reserved (8x)

14.10 Reserved (9x)

14.11 Reserved (Ax)

14.12 Firmware version (Bx)

B0: Request firmware version number. (Same as B1 01.)

B1 01: Request firmware version number.

B1 02: Request Bosch MCAN core release information (for CAN2 and CAN3).

B1 03: Request model number. (Same as old 'F0' command.)

B1 04: Request MAC address.

B1 05: LIN2-7 firmware version number.

B1 06: LIN2-7 model number.

B1 07: CAN2/3 firmware version number.

14.13 Reserved (Cx)

14.14 Reserved (Dx)

14.15 Reserved (Ex)

14.16 Model Query and Reset (Fx)

F0: Query for model number.

F1 A5: Restart the AVT-425 (a software reset).
This also resets the CAN2/3 microcontroller.
This also resets the LIN2-7 microcontroller.

F1 C3: Reset the Netburner CPU.
(This will cause an Ethernet disconnect.)

15 Responses

The first byte of all responses is as follows:

- High nibble, bits b7 - b4, indicates the response category.
- Low nibble, bits b3 – b0 indicates how many bytes are to follow.

The alternate header (1x) responses are the only exceptions to this format.

15.1 CAN / LIN / KWP Transmit acknowledgements (if enabled) (0x)

CAN0, CAN1, CAN2, CAN3
02 0r Az: Transmit ack.
r: channel number: 0, 1.
z: transmit object number.

CAN0, CAN1, CAN2, CAN3
06 jj kk ll mm 0r Az: Transmit ack.
jj kk ll mm: time stamp
r: channel number: 0, 1, 2, 3.
z: transmit object number, low nibble only.

LIN0, LIN1, KWP0, KWP1
02 0r pp: Transmit ack.
r: channel number: 7, 5, 8, 6.
pp: receive status byte (defined below).

LIN0, LIN1, KWP0, KWP1
06 jj kk ll mm 0r pp: Transmit ack.
jj kk ll mm: time stamp
r: channel number: 7, 5, 8, 6.

pp: receive status byte (defined below).

15.2 CAN / LIN / KWP Message received from the network (0x)

CAN0, CAN1
0x jj kk ll mm 0r qs tt vv ww zz nn pp ... :
x: count of bytes to follow.
jj kk ll mm: time stamp [optional]
r: channel: 0, 1.
q: b7: IDE.
0: 11-bit ID.
1: 29-bit ID.
b6: RTR.
0: normal frame.
1: RTR frame.
b5: 0
b4: 0
s: object number: $0 to $F.
tt vv: 11-bit ID, right justified.
tt vv ww zz: 29-bit ID, right justified.
nn pp...: data.

CAN2, CAN3
0x jj kk ll mm 0r qs tt vv ww zz nn pp ... :
x: count of bytes to follow.
jj kk ll mm: time stamp [optional]
r: channel: 2, 3.
q: b7: IDE.
0: 11-bit ID.
1: 29-bit ID.
b6: RTR.
0: normal frame.
1: RTR true, remote transmit request.
b5: FDF.
0: Classical CAN frame.
1: CAN-FD frame.
b4: BRS.
0: data field at normal speed.
1: data field at high speed.
s: object number: $0 to $F.
tt vv: 11-bit ID, right justified.
tt vv ww zz: 29-bit ID, right justified.
nn pp...: data.

LIN0, LIN1, and LIN2 thru LIN7
0x jj kk ll mm 0r ss id tt vv ww ... :
x: count of bytes to follow.
jj kk ll mm: time stamp [optional]
r: channel number: 7, 5, and A thru F.
ss: status byte (defined below).

id: message id.
tt vv ww … : data.

LIN status byte

b07: buffer closed by frame time out.
b06: from this device.
b05: from my periodic message (LIN2 thru 7).
b04: buffer closed by last byte timer.
b03: buffer opened without break.
b02: buffer closed due to max byte count.
b01: buffer closed by break.
b00: checksum error.

KWP0, KWP1
0x jj kk ll mm 0r ss tt vv ww ... :
x: count of bytes to follow.
jj kk ll mm: time stamp [optional]
r: channel number: 8, 6.
ss: status byte (defined below).
tt vv ww … : data.

KWP status byte

b07: buffer closed by last byte timer.
b06: from this device.
b05: 0.
b04: 0.
b03: 0.
b02: buffer closed due to max byte count.
b01: 0.
b00: checksum error.

15.3 CAN packet received from the network; alternate header formats (1x)

CAN0, CAN1
11 xx jj kk ll mm 0r qs tt vv ww zz nn pp... :
xx: count of bytes to follow.
jj kk ll mm: time stamp [optional]
r: channel: 0, 1.
q: b7: IDE.
    0: 11-bit ID.
    1: 29-bit ID.
  b6: RTR.
    0: normal frame.
    1: RTR frame.
  b5: 0
  b4: 0
s: object number.
tt vv: 11-bit ID, right justified.
tt vv ww zz: 29-bit ID, right justified.
nn pp ...: data.

CAN0, CAN1
12 xx yy jj kk ll mm 0r qs tt vv ww zz nn pp... :

xx yy :	count of bytes to follow.		
jj kk ll mm:	time stamp [optional]		
r:	channel: 0, 1.		
q:	b7:	IDE.	
		0:	11-bit ID.
		1:	29-bit ID.
	b6:	RTR.	
		0:	normal frame.
		1:	RTR frame.
	b5:	0	
	b4:		
s:	object number.		
tt vv:	11-bit ID, right justified.		
tt vv ww zz:	29-bit ID, right justified.		
nn pp ...:	data.		

CAN2, CAN3
11 xx jj kk ll mm 0r qs tt vv ww zz nn pp... :

xx:	count of bytes to follow.		
jj kk ll mm:	time stamp [optional]		
r:	channel: 2, 3.		
q:	b7:	IDE.	
		0:	11-bit ID.
		1:	29-bit ID.
	b6:	RTR.	
		0:	normal frame.
		1:	RTR true, remote transmit request.
	b5:	FDF.	
		0:	Classical CAN frame.
		1:	CAN-FD frame.
	b4:	BRS.	
		0:	data field at normal speed.
		1:	data field at high speed.
s:	object number: $0 to $F.		
tt vv:	11-bit ID, right justified.		
tt vv ww zz:	29-bit ID, right justified.		
nn pp ...:	data.		

CAN2, CAN3
12 xx yy jj kk ll mm 0r qs tt vv ww zz nn pp... :

xx yy :	count of bytes to follow.		
jj kk ll mm:	time stamp [optional]		
r:	channel: 2, 3.		
q:	b7:	IDE.	
		0:	11-bit ID.
		1:	29-bit ID.
	b6:	RTR.	

0: normal frame.
1: RTR true, remote transmit request.
b5: FDF.
0: Classical CAN frame.
1: CAN-FD frame.
b4: BRS.
0: data field at normal speed.
1: data field at high speed.
s: object number: $0 to $F.
tt vv: 11-bit ID, right justified.
tt vv ww zz: 29-bit ID, right justified.
nn pp ...: data.

CAN2, CAN3
Refer to the '11 bb 3r …' response for the long periodic message definition response.

15.4 Error Responses (2x)

21 01 Inbound command too long, flushed.

----
21 02 FIFO2 too full, flushed and reset.

----
22 03 01 FIFO1p1 overflow.
22 03 02 FIFO1p2 overflow.
22 03 03 FIFO1p3 overflow.
22 03 04 FIFO1p4 overflow.

----
22 04 xx Bad FIFO42_state variable.
xx: byte read from FIFO4.

----
23 05 xx yy DSPI channel 1 error flags.
xx yy: error flags.

----
25 06 xx xx xx xx ATSAM firmware application name does not start with 0x0425
xx xx xx xx: ATSAM firmware application name

----
21 07 FIFO2 overflow, FIFO was cleared and reset.

----
22 08 xx PIC protocol error: buff42 overrun. Buffer was cleared, and protocol was reset
xx: received byte

----
23 09 xx yy PIC protocol error: received packet length mismatch. Buffer was cleared, and protocol was reset.
xx: expected packet length

yy: received packet length

----
21 0A PIC protocol error: repeated STX condition. Buffer was cleared, and protocol was reset.

2x 0B

2x 0C

----
23 0D rr ss Error writing start-up parameters in non-volatile memory.
rr ss: write function return code.

----
22 0E xx ATSAM remained busy for longer than 15 us.
xx: number of ATSAM busy failures

----
21 0F ATSAM failed to start-up within 50 ms.

----
21 10 CAN0 initialization error.

----
21 11 CAN1 initialization error.

----
21 12 CAN2 initialization error.

----
21 13 CAN3 initialization error.

----
21 14 Long periodic message length invalid (See Long Periodic Message Definition And Response)

----
22 15 xx Long periodic message channel invalid (only available on CAN2 and CAN3)
xx: channel number. 0 = CAN2, 1 = CAN3, others invalid

----
23 16 xx yy CAN2 or CAN3 firmware reset failed.
xx: channel number. 2 = CAN2, 3 = CAN3
yy: low byte of response from ATSAM

----
22 17 xx Initialization: Nonvolatile user parameter settings do not match current ATSAM settings. ATSAM settings have been updated to match.
xx: 01 = Termination enable settings do not match
02 = Transceiver enable settings do not match

----
22 18 xx　　CAN dlc length error in 7x 18 periodic message command response.

----
21 19　　FIFO3 full after transfer of command.
in: "send_cmd_to_424".

----
23 1A yy zz　　UART0 initialization verification error.
yy: umr0.
zz: umr1.

----
22 1B ss UART1 error flags, bit map
b7: buffer1 was used
b6:
b5:
b4:
b3:
b2:
b1:
b0: no buffer available, byte discarded

----
22 1C ss UART2 error flags, bit map
b7: buffer1 was used
b6:
b5:
b4:
b3:
b2:
b1:
b0: no buffer available, byte discarded

----
22 1D xx　　"send_cmd_to_424" returned an error code.
xx: 0 = success
1 = invalid
2 = fifo3 full
3 = command too long

2x 1E

2x 1F

----
22 20 yy　　Command processing time out.

yy: header of offending command.

----
21 21 UART baud rate index error in 5x 50 command,

2x 22

----
23 23 0x 0y: Buffer index error in kwp_rcv_mgr.
x: channel index.
y: buffer index.

----
23 24 0x yy: One byte message in kwp_rcv_mgr.
x: channel number.
yy: the one byte.

----
24 25 0x 0y zz: Invalid buffer state in kwp_rcv_mgr.
x: channel number.
y: buffer index.
zz: buffer state.

----
24 26 0x yy zz: UART errors in kwp_new_byte.
x: channel number.
yy: receive status.
zz: receive data.

----
22 27 0x: No receive buffer available in kwp_new_byte.
x: channel number.

----
22 28 0x: Transmit command too short in kwp_xmt_cmd_proc.
x: channel number.

----
22 29 0x: Transmit command too long in kwp_xmt_cmd_proc.
x: channel number.

2x 2A

----
22 2B 0x: Invalid KWP transmit state.
x: channel number.

----

22 2C 0x: Loss of KWP arbitration limit. Message deleted.
x: channel number.

2x 2D

----
23 2E 0x ss: KWP transmit state watchdog expired.
x: channel number.
ss: transmit state.

2x 2F

2x 30

----
22 31 0x: UART short to ground detected.
x: channel number.

----
22 32 0x: 6x 13 command error, invalid channel number
x: channel number.

2x 33

----
22 34 yy: Read command time out.
yy: header of offending command.

2x 35

2x 36

----
22 37 0x: Fast Init manager, received byte error.
x: channel number.

2x 38

2x 39

----
23 3A rr ss: LIN0 lost frame report.

rr ss: count of lost frames.

----
23 3B rr ss: LIN1 lost frame report.
rr ss: count of lost frames.

----
23 3C rr ss: KWP0 lost frame report.
rr ss: count of lost frames.

----
23 3D rr ss: KWP1 lost frame report.
rr ss: count of lost frames.

----
23 3E rr ss: Buffer42 lost packet report
rr ss: count of lost packets.

2x 3F

2x 40

----
23 41 01 0x Operation mode error, UART1.
x: operation mode.
0 = LIN
1 = KWP
Others invalid.

----
23 41 02 0x Operation mode error, UART2.
x: operation mode.
0 = LIN
1 = KWP
Others invalid.

----
22 42 0x LIN Transmit index error, xmt_ix 'x'.

----
22 43 0x LIN Buffer index error, buff_ix 'x'.

2x 44

----
22 45 0x LIN channel 'x' transmit command too short.

----

22 46 0x LIN channel ‘x’ transmit command too long.

2x 47

2x 48

----

22 49 0x LIN channel 'x' uart transmit buffer not empty.

----

23 4A 0x yy 'ID byte only' message (LIN) / 'one byte only' message (KWP)
x: channel number.
yy: buffer state.

----

22 4B 0x LIN loss of arbitration.
x: channel number.

----

22 4C 0x LIN synch byte error.
x: channel number.

----

22 4D 0x LIN transmit ID byte error.
x: channel number.

----

22 4E 0x LIN invalid transmit state.
x: channel number.

----

22 4F 0x LIN transmit watchdog expired.
x: channel number.

2x 50

----

25 51 0x yy rr ss CAN channel x, transmit warning
x: channel number
yy: transmit counter
rr ss: error status register (low word) (ERRSTAT)

----

25 52 0x yy rr ss CAN channel x, receive warning
x: channel number
yy: receive counter

rr ss: error status register (low word) (ERRSTAT)

----
24 53 0x rr ss: CAN channel x, bit1 error
x: channel number
rr ss: error status register (low word) (ERRSTAT)

----
24 54 0x rr ss: CAN channel x, bit0 error
x: channel number
rr ss: error status register (low word) (ERRSTAT)

----
24 55 0x rr ss: CAN channel x, CRC error
x: channel number
rr ss: error status register (low word) (ERRSTAT)

----
24 56 0x rr ss: CAN channel x, framing error
x: channel number
rr ss: error status register (low word) (ERRSTAT)

----
24 57 0x rr ss: CAN channel x, stuff bit error
x: channel number
rr ss: error status register (low word) (ERRSTAT)

ERRSTAT bit definitions
bit15: BIT1 error
bit14: BIT0 error
bit13: ACK error
bit12: CRC error
bit11: FRAMING error
bit10: STUFF bit error
bit09: Transmit warning
bit08: Receive warning
bit07: IDLE
bit06: equals 1 if transmitting
bit05: fault confinement bit1
bit04: fault confinement bit 0
bit03: 0
bit02: bus-off interrupt flag
bit01: error interrupt flag
bit00: 0

fault confinement:
00 = can controller in error active state (normal)
01 = can controller in error passive state
1x = can controller in bus-off state

2x 58

2x 59

2x 5A

2x 5B

2x 5C

2x 5D

2x 5E

(All '2x 5F yy' error responses are related to ISO 15765 and only apply to CAN0 or CAN1.)

----
22 5F 01 canA_xmt_12: data count too long, 11-bit id, 'ae' disabled.

----
22 5F 02 canA_xmt_12: data count too long, 29-bit id, 'ae' disabled.

----
22 5F 03 canA_xmt_12: data count too long, 11-bit id, 'ae' enabled.

----
22 5F 04 canA_xmt_12: data count too long, 29-bit id, 'ae' enabled.

----
22 5F 05 canA_iso_xmt_buff_mgr: buff_state = 0x13, separation timer not expired, watchdog timeout.

----
22 5F 06 canA_iso_xmt_buff_mgr: buff_state = 0x13, object not available, watchdog timeout.

----
22 5F 07 canB_iso_xmt_buff_mgr: buff_state = 0x01, invalid object number.

----
22 5F 08 canA_iso_xmt_buff_mgr: buff_state = 0x14, watchdog timeout.

----
22 5F 09 canB_iso_xmt_buff_mgr: buff_state = 0x13, invalid object number.

----
22 5F 0A canA_iso_xmt_buff_mgr: invalid channel number.

----
22 5F 0B canA_iso_xmt_buff_mgr: invalid buffer number.

----
22 5F 0C canA_iso_xmt_buff_mgr: buff_state = 0x12, waiting for flow control frame, watchdog timeout.

----
22 5F 11 canA_iso_rcv_mgr: invalid channel number.

----
22 5F 12 canA_iso_rcv_mgr: invalid object number.

----
22 5F 13 canA_iso_rcv_mgr: frame dlc too long.

----
22 5F 14 canA_iso_rcv_mgr: frame dlc too short with 'ae'.

----
22 5F 15 canA_iso_rcv_mgr: frame dlc too short without 'ae'.

----
22 5F 16 canA_iso_rcv_mgr: single frame, byte count less than 'pci'.

----
22 5F 17 canA_iso_rcv_mgr: consecutive frame, invalid buffer number.

----
22 5F 18 canA_iso_rcv_mgr: unexpected frame sequence number.

----
22 5F 19 canA_iso_rcv_mgr: first frame, byte count zero.

----
22 5F 1A canA_iso_rcv_mgr: first frame, no buffer available.

----
22 5F 1B canA_iso_rcv_mgr: first frame, expected byte count of zero.

----

22 5F 1C canA_iso_rcv_mgr: flow control frame, byte count too short.

----
22 5F 1D canA_iso_rcv_mgr: flow control frame, this buffer not expecting a flow control frame.

----
22 5F 1E canA_iso_rcv_mgr: flow control frame, invalid separation time, 0x80 to 0xF0.

----
22 5F 1F canA_iso_rcv_mgr: flow control frame, invalid separation time, 0xFA to 0xFF.

----
22 5F 20 canA_iso_rcv_mgr: flow control frame, invalid flow status.

----
22 5F 21 canA_iso_rcv_mgr: unknown frame type, 'pci' byte upper nibble is unknown.

----
22 5F 22 canA_iso_rcv_mgr: invalid buffer state.

----
22 5F 23 canA_iso_xmt_buff_mgr: buffer state = 0x01, invalid object mate number.

----
22 5F 24 canA_iso_xmt_buff_mgr: buffer state = 0x11, invalid object number.

----
22 5F 25 canA_iso_xmt_buff_mgr: buffer state = 0x11, object not available, watchdog timeout.

----
22 5F 26 canA_iso_xmt_buff_mgr: buffer state = 0x13, invalid object number.

----
22 5F 31 canA_iso_buff_mgr: invalid channel number.

----
22 5F 32 canA_iso_buff_mgr: invalid buffer number.

----
22 5F 33 canA_iso_buff_mgr: invalid buffer state, 0x05 - 0x10 (inclusive).

----
22 5F 34 canA_iso_buff_mgr: invalid buffer state, 0x15 - 0xFF (inclusive).

----
22 5F 35 canA_iso_buff_mgr: invalid mate entry.

----
22 5F 36 canA_iso_buff_mgr: invalid buffer object entry.

----
22 5F 37 canA_iso_buff_mgr: invalid obj_buff entry.

----
22 5F 3A canA_iso_rcv_buff_mgr: invalid channel number.

----
22 5F 3B canA_iso_rcv_buff_mgr: invalid buffer number.

----
22 5F 3C canA_iso_rcv_buff_mgr: buffer state = 0x01, watchdog time-out.

----
22 5F 3D canA_iso_rcv_buff_mgr: buffer state = 0x02, watchdog time-out.

----
22 5F 3E canA_iso_rcv_buff_mgr: buffer state = 0x04, watchdog time-out.

----
22 5F 40

----
22 5F 41 canB_iso_rcv_mgr, invalid channel number.

----
22 5F 42 canB_iso_rcv_mgr, invalid object number.

----
23 5F 43 0y canB_iso_rcv_mgr, DLC > 8 and FDF not set.
'y' is CAN channel number.

----
23 5F 44 0y canB_iso_rcv_mgr, DLC too short, with ae.
'y' is CAN channel number.

----
23 5F 45 0y canB_iso_rcv_mgr, DLC too short, without ae.
'y' is CAN channel number.

----
23 5F 46 0y canB_iso_rcv_mgr, (frame_cnt < pci_cnt).
'y' is CAN channel number.

----
23 5F 47 xx canB_iso_rcv_mgr, invalid buffer number for consecutive frame.
'xx' = buffer number.

----
23 5F 48 0y canB_iso_rcv_mgr, invalid buffer state.
'y' is CAN channel number.

----
23 5F 49 0y canB_iso_rcv_mgr, invalid consecutive frame sequence number.
'y' is CAN channel number.

----
23 5F 4A 0y canB_iso_rcv_mgr, frame_cnt = zero in first frame.
'y' is CAN channel number.

----
23 5F 4B 0y canB_iso_rcv_mgr, pci_cnt = 0 in first frame.
'y' is CAN channel number.

----
23 5F 4C 0y canB_iso_rcv_mgr, pci_cnt > 8192 in first frame.
'y' is CAN channel number.

----
23 5F 4D 0y canB_iso_rcv_mgr, no buffer available.
'y' is CAN channel number.

----
23 5F 4E 0y canB_iso_rcv_mgr, flow control frame too short.
'y' is CAN channel number.

----
23 5F 4F 0y canB_iso_rcv_mgr, buffer not expecting flow control frame.
'y' is CAN channel number.

----
23 5F 50 0y canB_iso_rcv_mgr, invalid separation time in flow control frame, $80 to $F0.
'y' is CAN channel number.

----
23 5F 51 0y canB_iso_rcv_mgr, invalid separation time in flow control frame, > $FA.
Set to 1 msec.
'y' is CAN channel number.

----
23 5F 52 0y canB_iso_rcv_mgr,invalid flow status in flow control frame.
'y' is CAN channel number.

----
23 5F 53 0y canB_iso_rcv_mgr, unknown frame type.
'y' is CAN channel number.

----
22 5F 54 canB_iso_rcv_buff_mgr, invalid channel number.

----
22 5F 55 canB_iso_rcv_buff_mgr, invalid buffer number.

----
23 5F 56 0y canB_iso_rcv_buff_mgr, buffer time-out, first frame received,
flow control transmit pending.
'y' is CAN channel number.

----
23 5F 57 0y canB_iso_rcv_buff_mgr, buffer time-out while receiving data.
'y' is CAN channel number.

----
23 5F 58 0y canB_iso_rcv_buff_mgr, time-out waiting to send buffer to Client.
'y' is CAN channel number.

----
22 5F 59 canB_xmt_0x, RTR can not be true for ISO 15765 frame.

----
22 5F 5A canB_xmt_12, RTR can not be true for ISO frame.

----
22 5F 5B canB_iso_xmt_proc_FD, 11-bit ID, no ae, transmit command too short.

----
22 5F 5C canB_iso_xmt_proc_FD, 11-bit ID, no ae, transmit command too long.

----
22 5F 5D canB_iso_xmt_proc_FD, 11-bit ID, with ae, transmit command too short.

----
22 5F 5E canB_iso_xmt_proc_FD, 11-bit ID, with ae, transmit command too long.

----
22 5F 5F canB_iso_xmt_proc_FD, 29-bit ID, no ae, transmit command too short.

----
22 5F 60 canB_iso_xmt_proc_FD, 29-bit ID, no ae, transmit command too long.

----
22 5F 61 canB_iso_xmt_proc_FD, 29-bit ID, with ae, transmit command too short.

----
22 5F 62 canB_iso_xmt_proc_FD, 29-bit ID, with ae, transmit command too long.

2x 5F 63

2x 5F 64

----
22 5F 65 canB_iso_xmt_buff_mgr, invalid channel number.

----
22 5F 66 canB_iso_xmt_buff_mgr, invalid buffer number.

----
22 5F 67 canB_iso_xmt_buff_mgr, invalid object mate number.

----
24 5F 68 0y ss canB_iso_xmt_buff_mgr, watchdog expired.
'y' is CAN channel number.
'ss' is buffer state

2x 5F 69

----
22 5F 6A canB_iso_xmt_buff_mgr, buffer state = 13, byte count = 0.

2x 5F 6B

2x 5F 6C

2x 5F 6D

----
22 5F 6E canB_iso_buff_mgr, invalid channel number.

----
22 5F 6F canB_iso_buff_mgr, invalid buffer number.

----
23 5F 70 0y canB_iso_buff_mgr, invalid buffer state.
'y' is CAN channel number.

----
23 5F 71 0y canB_iso_buff_mgr, invalid buffer state.
'y' is CAN channel number.

----
23 5F 72 0y canB_iso_buff_mgr, invalid object number.

'y' is CAN channel number.

----
23 5F 73 0y canB_iso_buff_mgr, invalid mate entry.
'y' is CAN channel number.

2x 5F 74

----
22 5F 75 canB_iso_xmt_proc_FD, error in 'nearest dlc', single frame, type1, with ae, padding disabled.

----
22 5F 76 canB_iso_xmt_proc_FD, invalid channel number

----
22 5F 77 canB_iso_xmt_proc_FD, invalid object number

----
22 5F 78

----
22 5F 79 canB_iso_xmt_proc_CC, 11-bit command too short, no ae.

----
22 5F 7A canB_iso_xmt_proc_CC, 11-bit command too short, with ae.

----
22 5F 7B canB_iso_xmt_proc_CC, 29-bit command too short, no ae.

----
22 5F 7C canB_iso_xmt_proc_CC, 29-bit command too short, with ae.

----
22 5F 7D canB_iso_xmt_proc_CC, 11-bit command too long, no ae.

----
22 5F 7E canB_iso_xmt_proc_CC, 11-bit command too long, with ae.

----
22 5F 7F canB_iso_xmt_proc_CC, 29-bit command too long, no ae.

----
22 5F 80 canB_iso_xmt_proc_CC, 29-bit command too long, with ae.

----
23 5F 81 0y canB_iso_xmt_buff_mgr, bad dlc decode, buffer state = 13, no ae.
'y' is CAN channel number.

----
22 5F 82 canB_iso_xmt_buff_mgr, bad dlc decode, buffer state = 13, with ae.

----
23 5F 83 0y canB_iso_buff_mgr: invalid obj_buff entry.
'y' is CAN channel number.

----
22 5F 84

----
24 5F 85 xx xx canB_iso_xmt_proc_FD, no ae, too much data for buffer
xx xx: data count
----
24 5F 86 xx xx canB_iso_xmt_proc_FD, with ae, too much data for buffer
xx xx: data count
----
23 5F 87 xx canB_iso_xmt_proc_CC, invalid channel number
xx: channel number
----
23 5F 88 xx canB_iso_xmt_proc_CC, invalid object number
xx: object number

2x 60 00

----
22 60 01 AVT-424 Command too long, command flushed.

----
22 60 02 AVT-424 Command buffer mode error.

----
23 60 03 yy AVT-424 Command not processed.
yy: header byte of offending command.

----
23 60 04 yy AVT-424 Command processing failed.
yy: header byte of offending command.

----
23 60 05 yy AVT-424 No such transmit channel.
yy: header byte of offending command.

----
23 60 06 yy AVT-424 Command buffer watchdog timeout.
yy: header byte of offending command.

----
22 60 07: AVT-424 Error in timer update routine.

2x 61

Notes:
2x 62 ss … error responses are from channel LIN2.
2x 63 ss … error responses are from channel LIN3.
2x 64 ss … error responses are from channel LIN4.
2x 65 ss … error responses are from channel LIN5.
2x 66 ss … error responses are from channel LIN6.
2x 67 ss … error responses are from channel LIN7.

All 'ss …' codes are listed below.

----
24 6r 01 ss tt LINr error flags, flag byte follows
r: LIN channel number 2 thru 7
ss tt: bit map follows
b15:
b14:
b13:
b12:

b11:
b10: buffer1 invalid state > 0x06
b09: buffer1 invalid state
b08: buffer0 invalid state > 0x06

b07: buffer0 invalid state == 0
b06: rcvd byte, not break, no buffer available
b05: rcvd byte, not break, no active buffer open
b04: break byte not 00

b03: buff1 synch byte error
b02: buff0 synch byte error
b01: break rcvd, buff0 and buff1 not idle
b00: k-line short to ground detected

----
24 6r 02 ss tt UART error(s) detected with received byte
r: LIN channel number 2 thru 7
ss tt: bit map follows
b15: UTXISEL1 - xmt interrupt mode bit1
b14: UTXINV - xmt polarity
b13: UTXISEL0 - xmt interrupt mode bit0
b12: URXEN - rcv enable

b11: UTXBRK - xmt a break symbol

b10: UTXEN - xmt enable
b09: UTXBF - xmt buffer full
b08: TRMT - xmt shift register empty

b07: URXISEL1 - rcv interrupt mode bit1
b06: URXISEL0 - rcv interrupt mode bit0
b05: ADDEN - address character detect
b04: RIDLE - rcv idle

b03: PERR - parity error
b02: FERR - framing error
b01: OERR - rcv buffer overrun
b00: URXDA - rcv data available

----
23 6r 03 ss One byte message buffer0.
r: LIN channel number 2 thru 7
ss: received byte.

If timestamps are enabled:
27 6r 03 tt tt tt tt ss
r: LIN channel number 2 thru 7
tt tt tt tt: timestamp
ss: received byte

----
23 6r 04 ss One byte message buffer1.
r: LIN channel number 2 thru 7
ss: received byte.

If timestamps are enabled:
27 6r 04 tt tt tt tt ss
r: LIN channel number 2 thru 7
tt tt tt tt: timestamp
ss: received byte

----
22 6r 05 'xmt_type' error in transmit command processor.
r: LIN channel number 2 thru 7

----
22 6r 06 Transmit command too short, 12 xx yy.
r: LIN channel number 2 thru 7

----
22 6r 07 Transmit command too long, 12 xx yy, (do not add checksum).
r: LIN channel number 2 thru 7

----
22 6r 08 Transmit command too long, 12 xx yy, (do add checksum).

r: LIN channel number 2 thru 7

----

22 6r 09 Transmit command too short, slave.
r: LIN channel number 2 thru 7

----

22 6r 0A Transmit command too short, master.
r: LIN channel number 2 thru 7

----

22 6r 0B Transmit command too long, 0x, (do add checksum).
r: LIN channel number 2 thru 7

----

22 6r 0C Transmit command too long, 0x, (do not add checksum).
r: LIN channel number 2 thru 7

----

22 6r 0D Transmit buffer state == 0; invalid.
r: LIN channel number 2 thru 7

----

22 6r 0E Transmit buffer state > 0x03; invalid.
r: LIN channel number 2 thru 7

----

22 6r 0F Transmit buffer watchdog expired.
r: LIN channel number 2 thru 7

----

22 6r 10 not defined
r: LIN channel number 2 thru 7

----

22 6r 11 Received byte not equal to xmt byte; data byte.
r: LIN channel number 2 thru 7

----

22 6r 12 not defined
r: LIN channel number 2 thru 7

----

22 6r 13 not defined
r: LIN channel number 2 thru 7

----

22 6r 14 not defined
r: LIN channel number 2 thru 7

----
22 6r 15 Received byte not equal to xmt byte; synch byte.
r: LIN channel number 2 thru 7

----
22 6r 16 Received byte not equal to xmt byte; id byte only transmission.
r: LIN channel number 2 thru 7

----
22 6r 17 Received byte not equal to xmt byte; last data byte.
r: LIN channel number 2 thru 7

----
22 6r 18 Invalid transmit state in lin2_xmt_mgr.
r: LIN channel number 2 thru 7

----
22 6r 19 Invalid transmit state in lin2_xmt_mgr.
r: LIN channel number 2 thru 7

----
22 6r 1A Transmit state watchdog time out in lin2_xmt_mgr.
r: LIN channel number 2 thru 7

----
22 6r 1B UART transmit buffer full or shift register not empty.
r: LIN channel number 2 thru 7

----
22 6r 1C LIN channel not enabled, xmt command not processed.
r: LIN channel number 2 thru 7

----
22 6r 1D Transmit command, master/slave byte is neither.
r: LIN channel number 2 thru 7

----
22 6r 1E not defined
r: LIN channel number 2 thru 7

----
22 6r 1F not defined
r: LIN channel number 2 thru 7

----
22 6r 20 Stored pm length too long, in 7x 18 query.
r: LIN channel number 2 thru 7

----
22 6r 21 not defined

r: LIN channel number 2 thru 7

----
22 6r 22 not defined
r: LIN channel number 2 thru 7

----
22 6r 23 Invalid rcv buff0 state in lin2_new_byte.
r: LIN channel number 2 thru 7

----
22 6r 24 Invalid rcv buff1 state in lin2_new_byte.
r: LIN channel number 2 thru 7

----
22 6r 25 not defined
r: LIN channel number 2 thru 7

…

2x 70

…

----
23 7F 00 xx ‘0x’ transmit command, can2 or can3, invalid channel number
xx: channel number. 0 = can2, 1 = can3

----
23 7F 01 xx ’12’ transmit command, can2 or can3, invalid channel number
xx: channel number. 0 = can2, 1 = can3

----
22 7F 02

----
22 7F 03 Transmit command, invalid channel number.

----
22 7F 04 '12' transmit command, can0 or can1, object number byte,
bits 5:4 not zero or rtr bit not zero.

----
22 7F 05 '12' transmit command, can0 or can1, command too long, ISO 15765
not enabled.

----
22 7F 06 Transmit command, 11-bit, command too short.

----
22 7F 07 '0x' transmit command, 11-bit, command too long.

----
22 7F 08 Transmit command, 29-bit, command too short.

----
22 7F 09 '0x' transmit command, 29-bit, command too long.

----
22 7F 0A '12' transmit command, 11-bit ID, data length too long.
CAN2 or CAN3 only.

----
22 7F 0B '12' transmit command, 29-bit ID, data length too long.
CAN2 or CAN3 only.

----
22 7F 0C '12' transmit command, 11-bit ID, incorrect data length, padding disabled.
CAN2 or CAN3 only.

----
22 7F 0D '12' transmit command, 29-bit ID, incorrect data length, padding disabled.
CAN2 or CAN3 only.

----
22 7F 0E '12' transmit command; adl > dlc_lng.
CAN2 or CAN3 only.

----
22 7F 0F Transmit command, invalid bits in the object number byte.

----
22 7F 10 0x transmit command, 11-bit ID. Data length too short for DLC = 9 (12 data bytes) and padding disabled. CAN2 or CAN3 only.

----
22 7F 11 0x transmit command, 29-bit ID. Data length too short for DLC = 9 (12 data bytes) and padding disabled. CAN2 or CAN3 only.

----
22 7F 12 Improper 0x transmit command.
Can't have (FDF or BRS) true and RTR true.

----
22 7F 13 Improper 0x transmit command.

DLC > 8 and FDF flag is false.

----
22 7F 14 Improper ’12’ transmit command.
Can't have (FDF or BRS) true and RTR true.

----
22 7F 15 Improper ’12’ transmit command.
Can't have BRS true with FDF false.

----
22 7F 16 Improper ’12’ transmit command.
DLC > 8 and FDF flag is false.

----
22 7F 17 Improper 0x transmit command.
Can't have BRS true with FDF false.

----
22 7F 20 Object is receive busy in ’74 07’ command.

----
23 7F 21 yy canB_rcv_mgr invalid object number CAN2.
yy: object number

----
23 7F 22 yy canB_rcv_mgr invalid object number CAN3.
yy: object number

----
22 7F 23 canA_rcv_mgr invalid object number CAN0.

----
22 7F 24 canA_rcv_mgr invalid object number CAN1.

----
22 7F 25 canA_rcv_mgr invalid channel number.

----
22 7F 26 canA_rcv_mgr DLC too long.

----
22 7F 27 canA_rcv_mgr, a 'return(3)' from canA_iso_rcv_mgr.

----
22 7F 28 canA_error_mgr, invalid channel number.

----
22 7F 29 canA_xmt_ack_mgr, invalid channel number.

----
22 7F 2A canB_rcv_mgr, invalid channel number.

2x 7F 2B

----
22 7F 2C canB_error_mgr, invalid channel number.

----
23 7F 2D 0y canB_rcv_mgr, a 'return(3)' from canB_iso_rcv_mgr.
‘y’ is CAN channel number.

2x 7F 2E

2x 7F 2F

2x 7F 30

2x 7F 31

2x 7F 32

2x 80

2x 81

2x 82

2x 83

----
21 84 Command buffer mode fault.

2x 85

----
23 86 xx yy
LIN1 error flags.
b15:

b14:
b13:
b12:
b11:
b10:
b09:
b08:
b07: error in pit2 service code.
b06: illegal receive buffer state.
b05: break byte not 00.
b04: received a byte, not a break, no buffer available.
b03: received a byte, not a break, no active buffer.
b02: synch byte not $55.
b01: received byte errors: RB, FE, PE, OE.
b00: no receive buffer available.

2x 87

2x 88

2x 89

2x 8A

2x 8B

2x 8C

2x 8D

2x 8E

2x 8F

2x 90

----
24 91 0z aa bb

CANz error

z: CAN channel 2, 3.

aa bb: error flag bit map.
b15:
b14:
b13:
b12:
b11:
b10:
b9:
b8:
b7: bus off.
b6: bus warning.
b5: error counter ii overrun.
b4: FD protocol exception.
b3: RM protocol exception.
b2: transmit fifo overflow.
b1: receive fifo overflow.
b0: FDF is clear, dlc > 8.

----

22 92 0z
CANz bus off warning.
z: CAN channel 0, 1.

23 92 xx yy
ATSAM error flags
aa bb error flag bit map.
b15:
b14:
b13:
b12:
b11:
b10:
b09:
b08:
b07:
b06:
b05:
b04:
b03:
b02:
b01:
b00: invalid nb_addr in main loop

----

24 93 0z vv ww
CANz lost frame counter.
z: CAN channel 0, 1, 2, 3.
vv ww: lost frame counter.

2x 94

2x 95

----

23 96 xx yy

LIN0 error flags

b15:
b14:
b13:
b12:
b11:
b10:
b09:
b08:
b07: error in pit2 service code.
b06: illegal receive buffer state.
b05: break byte not 00.
b04: received a byte, not a break, no buffer available.
b03: received a byte, not a break, no active buffer.
b02: synch byte not $55.
b01: received byte errors: RB, FE, PE, OE.
b00: no receive buffer available.

2x 97

2x 98

2x 99

…

2x E5

2x E6

2x E7

2x E8

2x E9

15.5 Command error (3x)

31 yy		Command error.
	yy:	header of offending command.

----
32 yy FF		Command not processed.
	yy:	header of offending command.

----
32 xx yy		No such transmit channel number.
	xx:	header byte of offending command.
	yy:	channel number.

15.6 Reserved (4x)

15.7 Reserved (5x)

15.8 CAN / LIN / KWP Configuration reports (6x)

LIN0, LIN1, KWP0, KWP1, and LIN2 thru LIN7
63 01 0r 0z		Send received checksum to Client.
	r:	channel: 7, 5, 8, 6, and A thru F.
	z:	0:	disabled.
		1:	enabled.

LIN0, LIN1, KWP0, KWP1, and LIN2 thru LIN7
63 02 0r zz		Receiver buffer timeout.
	r:	channel: 7, 5, 8, 6, and A thru F.
	zz:	time in milliseconds.

LIN2 thru LIN7
62 04 00:		LIN2-7 microcontroller reset line is in the reset state.
62 04 01:		LIN2-7 microcontroller reset line in the run state.

Not channel specific
63 05 0r 0s:		Digital output status.
	r:	0:	output is high impedance.
		1:	output is low impedance.
	s:	0:	did NOT reset the time stamp counter.
		1:	did reset the time stamp counter.

CAN2, CAN3
63 06 0r 0s:		Response format status.
	r:	channel: 2 or 3.
	s:	0:	response format depends on message length.

1: always use the long (12 xx yy) format.

CAN0, CAN1, CAN2, CAN3, LIN0, LIN1, KWP0, KWP1, and LIN2 thru LIN7
63 08 0r 0y: Time stamp status.
r channel: 0, 1, 2, 3, 7, 5, 8, 6, and A thru F.
y: 0 disabled.
1 enabled.

LIN0, LIN1, KWP0, KWP1, and LIN2 thru LIN7
63 09 0r 0z: Selected bus pull-up resistor.
r: channel: 7, 5, 8, 6, and A thru F.
z: 0: 1 K ohm pull-up is enabled.
1: 31 K ohm pull-up is enabled.

LIN0, LIN1
64 1D 0r yy zz Synch break time.
r: channel: 7, 5.
yy zz: increment count. (increment = 1.024 usec).

KWP0, KWP1
64 27 0r zz P4. (P4 is transmit inter-byte time.)
r: channel: 8, 6.
zz: time in milliseconds.
[Default = 5 msec.]

KWP0, KWP1
64 28 0r 0z Format Byte processing status.
r: channel: 8, 6.
z: 0: disabled.
1: enabled.

KWP0, KWP1
63 2A 0r ss 'P3' – minimum inter-message time.
r: channel: 8, 6.
ss: time in milliseconds.

Not channel specific
64 31 0y 0r 0s: Set TCP parameters.
Note: The parameters are not listed here.
The Client / User should not use this command.

CAN0, CAN1, CAN2, CAN3, LIN0, LIN1, KWP0, KWP1, and LIN2 thru LIN7

63 40 0r 0y: Send transmit acknowledgements to Client.
r channel: 0, 1, 2, 3, 7, 5, 8, 6, and A thru F.
y: 0 disabled.
1 enabled.

KWP0, KWP1
64 46 0r ss tt Fast Init 'W5' K-line idle time.
r: channel: 8, 6.
ss tt: time in milliseconds.

KWP0, KWP1
63 47 0r ss Fast Init K-line low time.
r: channel: 8, 6.
ss: time in milliseconds.

KWP0, KWP1
63 48 0r ss Fast Init K-line high time.
r: channel: 8, 6.
ss: time in milliseconds.

KWP0, KWP1
63 4B 0r 0z KWP checksum method.
r: channel: 8, 6.
z: 0: no transmit checksum.
1: sum of bytes.
2: sum of bytes, 2's complement.
3: XOR of bytes.

LIN0, LIN1, KWP0, KWP1, and LIN2 thru LIN7
63 50 0r 0z Baud rate.
r: channel: 7, 5, 8 6, and A thru F.
z: 1: 2400 baud.
2: 9600 baud.
3: 19200 baud.
4: 10400 baud.

64 50 0r yy zz Baud rate.
r: channel: 7, 5, 8 6. (Not available for channels A thru F.)
yy zz: divisor load.
125,000,000 / 32 / yy zz
(where 'yy zz' is converted to decimal)

LIN0, LIN1, and LIN2 thru LIN7
63 52 0r zz Maximum frame time.

r: channel: 7, 5, and A thru F.
zz: time in milliseconds.

LIN0, LIN1, and LIN2 thru LIN7
63 5A 0r 0z Checksum type.
r: channel: 7, 5, and A thru F.
z: 0: classic (LIN 1.3).
1: enhanced (LIN 2.0).
2: none.

LIN0, LIN1, and LIN2 thru LIN7
63 66 0r 0z "ID byte only" error message to Client.
r: channel: 7, 5, and A thru F.
z: 0: disabled. [Default]
1: enabled.

KWP0, KWP1
63 66 0r 0z Send "One byte only" error message to Client.
r: channel numbers: 8, 6.
z: 0: disabled. [Default.]
1: enabled.

LIN0, LIN1, KWP0, KWP1, and LIN2 thru LIN7
63 69 0r 0z Secondary operations.
r: channel: 7, 5, 8, 6, and A thru F.
z: 0: disabled.
1: enabled.

Not channel specific
63 6A yy zz: Netburner: CPU heart beat LED blink rate.
yy zz: LED half period time, msec.

Not channel specific
63 6B yy zz: LIN2-7: CPU heart beat LED blink rate.
yy zz: LED half period time, msec.

Not channel specific
63 6C yy zz: CAN2/3: CPU heart beat LED blink rate.
yy zz: LED half period time, msec.
[Default = $01F4 => 500 msec.]

LIN0, LIN1, KWP0, KWP1, and LIN2 thru LIN7
64 7E 0r yy zz: Short to ground counter reset value.

r: channel: 7, 5, 8, 6, and A thru F.
yy zz: counter reset value.
[Default = $0100 => 256.]

----
not channel specific
65 80 rr ss tt vv: Stored (non-volatile) start-up parameters.
rr ss tt vv is the following bit map.

b31:	‘1’	reserved, not used.	
b30:	‘1’	reserved, not used.	
b29:	‘1’	reserved, not used.	
b28:	‘1’	reserved, not used.	
b27:	‘1’	reserved, not used.	
b26:	‘1’	reserved, not used.	
b25:	‘1’	reserved, not used.	
b24:	‘1’	reserved, not used.	
b23:	‘1’	reserved, not used.	
b22:	‘1’	reserved, not used.	
b21:	‘1’	reserved, not used.	
b20:	‘1’	reserved, not used.	
b19:	‘1’	reserved, not used.	
b18:	‘1’	reserved, not used.	
b17:	‘1’	reserved, not used.	
b16:	‘1’	reserved, not used.	
b15:	‘1’	reserved, not used.	
b14:	‘1’	reserved, not used.	
b13:	‘1’	reserved, not used.	
b12:	‘1’	reserved, not used.	
b11:	‘1’	reserved, not used.	
b10:	‘1’	reserved, not used.	
b09:	‘1’	reserved, not used.	
b08:	‘1’	reserved, not used.	
b07:	CAN3 transceiver control		1 = transceiver enabled. 0 = transceiver disabled.
b06:	CAN2 transceiver control		1 = transceiver enabled. 0 = transceiver disabled.
b05:	CAN1 transceiver control		1 = transceiver enabled. 0 = transceiver disabled.
b04:	CAN0 transceiver control		1 = transceiver enabled. 0 = transceiver disabled.
b03:	CAN3 termination		1 = termination enabled. 0 = termination disabled.
b02:	CAN2 termination		1 = termination enabled. 0 = termination disabled.
b01:	CAN1 termination		1 = termination enabled. 0 = termination disabled.
b00:	CAN0 termination		1 = termination enabled. 0 = termination disabled.

6F 81 01 ss ss ss ss ss ss ss ss ss ss ss ss ss: Stored (non-volatile) serial number.

ss…ss: serial number ASCII characters.
If less than 13 characters, the remaining bytes will be 0x00

15.9 KWP Initialization attempt responses (7x)

KWP0, KWP1
73 13 0r ss 'Fast Init' (ISO 14230) responses
r: channel number: 8, 6.
ss: 00 – init attempt failed.
01 – init sequence started.
11 – init sequence completed successfully.

15.10 CAN / LIN / KWP configuration reports (8x)

CAN2, CAN3
84 01 0r xx yy: The ISO15765 buffer time out reset value.
r: channel: 2, 3.
xx yy: reset value (msec).

The Client / User should NOT modify this value
The command should only be used for testing and debugging.

CAN2, CAN3
83 03 xx yy: The number of 1 usec wait intervals.
xx yy: count of wait intervals.

The Client / User should NOT modify this value
The command should only be used for testing and debugging.

CAN0, CAN1, CAN2, CAN3
84 04 0r 0z 0y Configuration of CAN object.
r channel: 0, 1, 2, 3.
z: object number: $0 to $F.
y: 0: object disabled.
1: object enabled for receive.
2: object enabled for transmit. (CAN0/1 only)

CAN0, CAN1
84 07 0r 0z 0w: Object transmit status.
z: object number: $0 to $F.
w: 0: transmission inactive.
1: object set to transmit.

CAN2, CAN3
82 09 yy: The number of received frames processed before exiting the receive manager.
yy: count.

The Client / User should NOT modify this value
The command should only be used for testing and debugging.

CAN0, CAN1
83 0A 0r yy: Baud rate.
r channel: 0, 1.
yy: 00: Client specified using 74 0B 0x rr ss command.
01: 1 Mbps.
02: 500 Kbps.
03: 250 Kbps.
04: 125 Kbps.
0A: 33.333 Kbps.
0B: 83.333 Kbps.

CAN2, CAN3
84 0A 0r yy zz: Baud rate.
r: channel: 2, 3.
slow baud rate
yy: 00: Client specified using 74 0B 0x rr ss command.
01: 1 Mbps.
02: 500 Kbps.
03: 250 Kbps.
04: 125 Kbps.
0A: 33.333 Kbps.
0B: 83.333 Kbps.
fast baud rate
zz: 00: Client specified using 74 0B 0x rr ss command.
01: 1 Mbps.
02: 500 Kbps.
03: 250 Kbps.
04: 125 Kbps.
0A: 33.333 Kbps.
0B: 83.333 Kbps.
0C: 2 Mbps.
0D: 4 Mbps.
0E: 5 Mbps.
0F: 8 Mbps.

CAN0, CAN1, CAN2, CAN3
86 0B 0r ss tt vv ww: Bit Timing Registers (BTR).
r: channel: 0, 1, 2, 3.
rr ss: Bit Timing Register 0.
vv ww: Bit Timing Register 1.

Refer to the '76 0B' command for information about register definitions.

CAN0, CAN1, CAN2, CAN3

83 0E 0r zz: ISO 15765 outbound flow control separation time.
- r: channel: 0, 1, 2, 3.
- zz: separation time used in an outbound flow control frame. time in milliseconds.

CAN0, CAN1, CAN2, CAN3

83 11 0r 0z: Operation state.
- r: channel: 0, 1, 2, 3.
- z: 0: disabled.
  - 1: enabled for normal operations.

CAN1

82 12 0y: SWC transceiver mode. CAN1 only.
- y: 0: Sleep mode.
  - 1: High speed mode.
  - 2: Wake up mode.
  - 3: Normal mode.

CAN0, CAN1, CAN2, CAN3

83 13 0r 0y: Operational status of CAN transceiver, 2-wire.
- r: channel: 0, 1, 2, 3.
- y: 0 – transceiver disabled.
  - 1 – transceiver enabled. [Default]

CAN2, CAN3 (Transmit object only.)

85 17 0r wz tt vv Transmit object configuration.
- r: CAN channel: 2, 3.
- w: b7: 0
  - b6: 0
  - b5: FDF
    - 0 = Classical CAN frame.
    - 1 = CAN-FD frame.
  - b4: BRS
    - 0 = data at regular speed.
    - 1 = data at high speed.
- z: object number: $0 to $F.
- tt vv: 11-bit ID, right justified.

87 17 0r wz tt vv mm nn Transmit object configuration.
- r: CAN channel: 2, 3.
- w: b7: 0
  - b6: 0
  - b5: FDF
    - 0 = Classical CAN frame.

1 = CAN-FD frame.
b4: BRS
0 = data at regular speed.
1 = data at high speed.
z: object number: $0 to $F.
tt vv mm nn: 29-bit ID, right justified.

CAN0, CAN1
8x 18 yr pp tt vv ww zz mm nn ... Periodic message set-up.
y: b7: IDE.
0: 11-bit ID.
1: 29-bit ID.
b6: RTR
0: normal frame.
1: RTR true, remote transmit request.
b5: 0
b4: 0
r: channel: 0, 1.
pp: message number: $00 to $2F.
tt vv: 11-bit ID, right justified.
tt vv ww zz: 29-bit ID, right justified.
mm nn ...: data field.

CAN2, CAN3
8x 18 yr pp tt vv ww zz mm nn ... Periodic message set-up.
y: b7: IDE.
0: 11-bit ID.
1: 29-bit ID.
b6: RTR. (only valid for Classical CAN frame).
0: normal frame.
1: RTR true, remote transmit request.
b5: FDF
0 = Classical CAN frame.
1 = CAN-FD frame.
b4: BRS
0 = data at regular speed.
1 = data at high speed.
r: channel: 2, 3.
pp: message number: $00 to $2F.
tt vv: 11-bit ID, right justified.
tt vv ww zz: 29-bit ID, right justified.
mm nn ...: data field.

CAN2, CAN3
11 bb 3r y0 pp tt vv ww zz mm nn ...
11: long header.
bb: count of bytes to follow.
3: long periodic message response.
r: channel number: 2, 3.

y: b7: IDE
1: 29-bit.
0: 11-bit.
b6: RTR
1: is an RTR frame.
0: is NOT an RTR frame.
b5: FDF
0 = Classical CAN frame.
1 = CAN-FD frame.
b4: BRS
0 = data at regular speed.
1 = data at high speed.
0
pp: message number: $00 to $2F.
tt vv: 11-bit id.
tt vv ww zz: 29-bit id.
mm nn ... : data field (0 to 64 bytes, inclusive).

LIN0, LIN1, and LIN2 thru LIN7
8x 18 0r 0p 0m rr ss tt … Periodic message set-up.
r: channel: 7, 5, and A thru F.
p: message number: $0 to $F.
m: 0: slave.
1: master.
1: master.
rr ss tt ...: data field. [optional].

KWP0, KWP1
8x 18 0r 0p ss tt vv… Periodic message set-up.
r: channel: 8, 6.
p: message number: $0 to $F.
ss tt vv...: data field.

CAN0, CAN1
84 19 0r pp 0y Object assigned to CAN periodic message.
r: Channel: 0, 1.
pp: Message number: $00 to $2F.
y: Object number: $0 to $F.

CAN0, CAN1, CAN2, CAN3
84 1A 0r pp 0v: Periodic message disable/enable status.
r: channel: 0, 1, 2, 3.
pp: message number: $00 to $2F.
v: 0 disabled.
1 enabled.

LIN0, LIN1, KWP0, KWP1, and LIN2 thru LIN7
84 1A 0r 0p 0v: Periodic message disable/enable.

r: channel: 7, 5, 8, 6, and A thru F.
p: message number: $0 to $F.
v: 0 disabled.
1 enabled.

CAN0, CAN1, CAN2, CAN3
85 1B 0r pp vv ww: Periodic message interval count.
r: channel: 0, 1, 2, 3.
pp: message number: $00 to $2F.
ww vv: interval count.

LIN0, LIN1, KWP0, KWP1, and LIN2 thru LIN7
85 1B 0r 0p vv ww: Periodic message interval count.
r: channel: 7, 5, 8, 6, and A thru F.
p: message number: $0 to $F.
vv ww: time in milliseconds.

CAN0, CAN1, CAN2, CAN3, LIN0, LIN1, KWP0, KWP1, and LIN2 thru LIN7
82 1C 0r All periodic message for channel 'r' disabled.
r: channel: 0, 1, 2, 3, 7, 5, 8, 6, and A thru F.
r: 'FF' – all messages, all channels disabled.

CAN2, CAN3
83 1F 0r 00: Do not process non-ISO15765 received frames.
83 1F 0r 01: Do process non-ISO15765 received frames.
r: channel: 2, 3.

CAN0, CAN1, CAN2, CAN3
83 25 0r ss: The flow control additional separation time.
r: channel: 0, 1, 2, 3.
ss: additional time, in milliseconds,
to add to the responding node separation time.

CAN0, CAN1, CAN2, CAN3
83 27 0r 0z 00: ISO 15765 padding is disabled.
r: channel: 0, 1, 2, 3.
z: object number: $0 to $F.
00: disabled.

84 27 0r 0z 01 ww: ISO 15765 padding is enabled.
r: channel: 0, 1, 2, 3.
z: object number: $0 to $F.
01: enabled.
ww: pad byte.

CAN0, CAN1, CAN2, CAN3

84 28 0r 0y 0s: Objects are paired; ISO 15765 operations are enabled. 'AE' disabled.
- r: channel: 0, 1, 2, 3.
- y: object number: $0 to $F.
- s: object number: $0 to $F.

85 28 0r 0y 0s ww: Objects are paired; ISO 15765 operations are enabled. 'AE' enabled.
- r: channel: 0, 1, 2, 3.
- y: object number: $0 to $F.
- s: object number: $0 to $F.
- ww: 'AE' byte.

CAN2, CAN3

84 29 0r 0y ss: 'max_dlc'. (Only used with ISO15765 processing.)
- r: channel: 2, 3.
- y: object number: $0 to $F.
- ss: only the following values are valid (hex digits):
  08, 0C, 10, 14, 18, 20, 30, 40.

CAN0, CAN1

85 2A 0r yz ss tt: Object 11-bit ID.
- r: channel: 0, 1.
- y: b7: 0.
  - b6: RTR.
    - 0: normal frame.
    - 1: RTR true, remote transmit request.
  - b5: 0
  - b4: 0
- z: object number: $0 to $F.
- ss tt: 11-bit ID.

87 2A 0r yz ss tt vv ww: Object 29-bit ID.
- r: channel: 0, 1.
- y: b7: 0.
  - b6: RTR.
    - 0: normal frame.
    - 1: RTR true, remote transmit request.
  - b5: 0
  - b4: 0
- z: object number: $0 to $F.
- ss tt vv ww: 29-bit ID.

CAN2, CAN3

85 2A 0r yz ss tt: Object 11-bit ID.
- r: channel: 2, 3.
- y: b7: 0.

b6: RTR.
0: normal frame.
1: RTR true, remote transmit request.
b5: 0
b4: 0
z: object number: $0 to $F.
ss tt: 11-bit ID.

87 2A 0r yz ss tt vv ww: Object 29-bit ID.
r: channel: 2, 3.
y: b7: 0.
b6: RTR.
0: normal frame.
1: RTR true, remote transmit request.
b5: 0
b4: 0
z: object number: $0 to $F.
ss tt vv ww: 29-bit ID.

CAN0, CAN1, CAN2, CAN3
85 2C 0r 0z ss tt: 11-bit mask.
r: channel: 0, 1, 2, 3.
z: mask number: $0 to $F.
ss tt: mask value, 11-bit.

87 2C 0r 0z ss tt vv ww: 29-bit mask.
r: channel: 0, 1, 2, 3.
z: mask number: $0 to $F.
ss tt vv ww: mask value, 29-bit.

1: bit must match.
0: bit is don't care.

CAN0, CAN1, CAN2, CAN3
84 30 0r 0z 00: ISO 15765 'AE' operation disabled.
r: channel: 0, 1, 2, 3.
z: object number: $0 to $F.

85 30 0r 0z 01 ww: ISO 15765 'AE' operation enabled.
r: channel: 0, 1, 2, 3.
z: object number: $0 to $F.
ww: 'ae' byte.

CAN0, CAN1, CAN2, CAN3, LIN0 thru LIN7
87 40 00 0r zz ww BB 0b: Rolling Counter Field Parameters for Periodic Message
r: channel: 0, 1, 2, 3, 5, 7, $A thru $F.
zz: message number: $00 to $2F.
ww: rolling counter width: $00 to $20

BB: byte number of the counter lsb: $00 to $3F
b: bit number of the counter lsb: $0 to $7

88 40 01 0r zz VV VV VV VV: Rolling Counter Value for Periodic Message
r: channel: 0, 1, 2, 3, 5, 7, $A thru $F.
zz: message number: $00 to $2F.
VV VV VV VV: 32-bit unsigned counter value

88 40 02 0r zz II II II II: Rolling Counter Increment/Decrement Value for Periodic Message
r: channel: 0, 1, 2, 3, 5, 7, $A thru $F.
zz: message number: $00 to $2F.
II II II II: 32-bit signed increment/decrement value

88 40 03 0r zz MM MM MM MM: Rolling Counter Maximum Value for Periodic Message
r: channel: 0, 1, 2, 3, 5, 7, $A thru $F.
zz: message number: $00 to $2F.
MM MM MM MM: 32-bit unsigned maximum value

85 40 04 0r zz 0e: Rolling Counter for Periodic Message Enable State
r: channel: 0, 1, 2, 3, 5, 7, $A thru $F.
zz: message number: $00 to $2F.
e: enable. 0 = disabled, 1 = enabled.

CAN0, CAN1, CAN2, CAN3, LIN0 thru LIN7
87 41 00 0r zz TT BB CC: Checksum Settings Parameters for Periodic Message
r: channel: 0, 1, 2, 3, 5, 7, $A thru $F.
zz: message number: $00 to $2F.
TT: checksum type:
0 = SAE_J1850
1 = SAE_J1850_ZERO
2 = HALLA_3346
BB: byte number of the checksum lsb: $00 to $3F
CC: message byte count for the checksum: $00 to $3F

75 40 04 0r zz 0e: Checksum for Periodic Message Enable/Disable State
r: channel: 0, 1, 2, 3, 5, 7, $A thru $F.
zz: message number: $00 to $2F.
e: enable. 0 = disabled, 1 = enabled.

CAN1
82 45 01: CAN1 transceiver to single wire CAN (SWC).
82 45 02: CAN1 transceiver to 2-wire CAN.

CAN2, CAN3
83 60 0r 0y: Status of extended data length padding.
r: channel: 2, 3.
y: 0: disabled.
1: enabled.

CAN2, CAN3
83 61 0r yy: Pad byte value.
r: channel: 2, 3.
yy: pad byte value.

CAN0, CAN1, CAN2, CAN3
83 62 0r 0y: CAN 2-wire bus termination state.
r: channel: 0, 1, 2, 3.
y: 0: disabled.
1 enabled.

15.11 Board status information (9x)

92 01 10: CAN0 reset complete.

92 01 11: CAN1 reset complete.

92 01 12: CAN2 (hardware or software) reset complete.

92 01 13: CAN3 (hardware or software) reset complete.

92 01 14: CAN2/3 hardware reset complete.

92 01 20: LIN0 reset complete.

92 01 21: LIN1 reset complete.

92 01 30: LIN2-7 – hardware reset complete.
(This report is generated by the AVT-425 CPU.)

92 01 31: LIN2-7 – firmware reset complete.

93 04 xx yy: Netburner firmware version report. Version is xx yy.

93 05 xx yy: LIN2-7 – firmware version report. Version is xx yy.

93 06 xx yy: LIN2-7 – board model number. AVT-425: xx yy = 04 25

93 07 xx yy: CAN2/3 – firmware version report. Version is xx yy.

91 0A: Power-on and 'F1 C3' reset.

91 0F: Response to an 'F1 A5' reset.

93 28 0x yz: Model number report. xyz is the model number.

91 3A: AVT-425 Ethernet connect response.

95 3B xx yy zz: CAN2/3 MCAN core release information.

97 3C pp rr ss tt vv ww: AVT-425 MAC address.

15.12 Reserved (Ax)

15.13 Reserved (Bx)

15.14 Reserved (Cx)

15.15 Reserved (Dx)

15.16 Reserved (Ex)

15.17 Reserved (Fx)

16 Questions and Contact

16.1 Contact Information

Contact Orion Measurement Solutions by e-mail or phone.

Address:
Orion Measurement Solutions
2704 Paldan Dr.
Auburn Hills, MI 48326 USA
Attn: Support

Phone: +1-248-475-9880
Support E-mail: support@orionmeas.com
Sales E-mail: sales@orionmeas.com
Web site: https://orionmeas.com/

16.2 Software tools:

AVT-425 Diagnostic Tool: OMS AVT Diag Tool
Set IP address (packaged with Diag Tool): Netburner OMS IP Setup
Update Operation Software (packaged with Diag Tool): AutoUpdate

https://orionmeas.com/

16.3 Firmware

The AVT-425 uses the following firmware files:

Netburner Operation Firmware file.
ATSAM Operation Firmware file.
PIC Operation Firmware file.

https://orionmeas.com/