ADVANCED VEHICLE TECHNOLOGIES, Inc.

AVT - 423

Multiple Interface

CAN0: 2-wire (high speed) (non-FD)

CAN1: 2-wire (high speed) or Single Wire (SWC) (non-FD)

CAN2: 2-wire (high speed) (FD capable)

CAN3: 2-wire (high speed) (FD capable)

LIN0: LIN communications

LIN1: LIN communications

LIN1: K-Line communications
(not functional, yet)

Flexray: Dual or Single channel communications
(not installed, not functional)

Firmware Version 00 25
13 January 2019

1509 Manor View Road, Davidsonville, MD 21035 USA
+1-410-798-4038 (voice) support@AVT-HQ.com
www.AVT-HQ.com

Table of Contents

1. Introduction

This document describes the AVT-423 hardware and firmware.

The AVT-423 is a multiple network interface for in-vehicle networks. The operation firmware supports the following networks/protocols on the indicated channels:

- CAN0: 2-wire CAN (non-FD).
  channel 0
- CAN1: 2-wire CAN or Single Wire CAN (SWC) (non-FD).
  channel 1
- CAN2: 2-wire CAN (CAN-FD capable).
  channel 2
- CAN3: 2-wire CAN (CAN-FD capable).
  channel 3
- LIN0: LIN only.
  channel 7
- LIN1; LIN or KWP (K-line).
  channel 5
- Flexray: Dual or Single channel. (Not yet functional.)
  channel 9

All operations are simultaneous with the exception of LIN or K-line on channel LIN1.

1.1 New Stuff

Starting with manual version '0023 A' I will try to put all new and updated text in 'blue'.

'Blue' stuff will indicate changes from the previous version of the manual.

1.2 Hardware

Refer to the web site for the most up-to-date information about board hardware status.

Hardware status: www.AVT-HQ.com/423_hw.htm

1.3 Firmware Status

At this time, the AVT-423 firmware supports the following capabilities.

1.3.1 CAN0

CAN (non-FD). Transmit and receive. Sixteen objects (either receive or transmit). Forty eight (48) periodic messages. Periodic messages are Type1 only.

1.3.2 CAN1

CAN (non-FD). Transmit and receive. Sixteen objects (either receive or transmit). Forty eight (48) periodic messages. Periodic messages are Type1 only.

1.3.3 CAN2

CAN and CAN-FD capable. Transmit and receive. Sixteen receive objects. Forty eight (48) periodic messages. Periodic messages are Type1 only. ISO CAN frame CRC (non-ISO is available). Maximum data payload of 64 bytes supported. Maximum baud rate of 8 Mbaud supported.

1.3.4 CAN3

CAN and CAN-FD capable. Transmit and receive. Sixteen receive objects. Forty eight (48) periodic messages. Periodic messages are Type1 only. ISO CAN frame CRC (non-ISO is available). Maximum data payload of 64 bytes supported. Maximum baud rate of 8 Mbaud supported.

1.4 Firmware Note #1

Firmware version '0012' includes significant changes to the following CAN configuration and operation commands (that existed in previous versions.

0x transmit command.

11 xx transmit command.

12 yy xx transmit command.

7x 2A configuration command.

More specifically, what changed is that the upper nibble bits of the channel byte have been moved to the object byte. The bits that have been moved include: RTR, IDE, EDL, and BRS.

Those bits were moved because they are associated to an 'object' not to a 'channel'. Therefore, it makes more sense for those control bits to be in the object byte of the command.

As with any rule, there are exceptions. The periodic message setup command (7x 18) still maintains those bits in the channel byte. This is because if they were moved to the periodic message byte, there would not be any room left in that byte should the need ever arise to increase the number of periodic messages available to each CAN channel.

1.5 Firmware Note #2

When implementing ISO 15765 capability into firmware version '0012' – it was deemed desirable to try and make the AVT-423 command set 'look' like the AVT-718 command set; as much as was reasonably possible. As it turned out, the new '7x 05' command is now identical to the redesigned '7x 2A' command. Several other commands taken from the AVT-718 have been implemented even though the user may never find them unnecessary. This was all done in an effort to make the AVT-423 command set 'friendly' to users familiar with the AVT-718.

Likwise, initial AVT-423 command development was focused on being 'friendly' to users familiar with the AVT-853 command set.

1.6 Firmware Note #3

Firmware version 0020 introduced support for ISO 15765 message formatting and handling for channels CAN2 and CAN3. The user should read Section xxx for information about setting up and using this new feature. ISO 15765 message formatting and handling for CAN-FD is significantly more complicated than that for Classical CAN.

The user "should" now include an object number in the transmit command for channels CAN2 and CAN3.

1.7 Firmware Plan

Firmware for this product is under almost continuous development. This manual will be updated as soon as possible for each new firmware release.

Future releases are expected to be for corrections and implementation of functions and/or capabilities as dictated by customer needs.

1.8 Firmware

Refer to the web site for the most up-to-date information about AVT-423 firmware versions:

www.AVT-HQ.com/432_sw.htm

1.8.1 Determining Firmware Version

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

- Connect to a client computer running the Hex Terminal or equivalent.
- Power-on the AVT-423 interface unit.
- The power-on notification is:
  $91 $3A indicates AVT-423 operations.
  $93 $04 $xx $yy where 'xx yy' is the firmware version.
- At any time, send the $B0 command.
- The response will be: $93 $04 $xx $yy where 'xx yy' is the firmware version.

1.8.2 Determining Model Number

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

- Connect to a client computer running the Hex Terminal or equivalent.
- Power on the AVT-423 interface unit.
- Send the $F0 command.
- The response will be: $93 $28 $xx $yy where xxyy forms the model number. (eg. 04 23)

1.8.3 Determining Board Revision Level

There are two revision levels in this product family.

- The "Circuit Configuration" revision level is written on the top (component side) of the PC board, in black marker, in the white "rev" block.
- "Board" revision level. This can be determined by looking at the bottom of the PC board (not the component side). Written in copper is the board revision level and date.

Commands and Responses

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

2. Glossary

Common terms, abbreviations, acronyms, and more.

$ sign	Indicates a hex number.
0x1234	Indicates hex number 1234. (I try to NOT to use this format here as it can be confused with other uses of '0x'.)
ADC	Analog to Digital Converter or Conversion.
BRS	A CAN bit. Baud rate switch. Part of CAN-FD format. Signals that the data portion of the CAN frame 'might' use a different baud rate.
CAN	Controller Area Network (aka: non-FD CAN).
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
EDL	A CAN bit. Extended Data Length. 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	An ISO specification for CAN and 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. This is a 'superset' of ISO 9141 and ISO 9141-2.
LIN	Local Interconnect Network.
LIN0	LIN, channel 7
LIN1	LIN, channel 5

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-423 Operation

The AVT-423 does not have a power switch. The unit powers up and begins operations as soon as external power is applied.

On power-up, the interface will, very quickly, report to the client computer: $91 $3A which indicates that the interface is an AVT-423.

The interface will then report $93 $04 $xx $yy where 'xx yy' is the unit firmware version number.

Note that the client computer can not establish a TCP/IP connection until the AVT-423 is fully operational. From power-on to full operation is about 5 seconds.

4. AVT-423 CPU

The AVT-423 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

AVT offers two PC applications to the user.
Both were supplied by Netburner.
Both are available for download from AVT's web site.
Refer to the last page of this manual for direct link to the web page.

Both are small PC applications (executable) that do not need to be installed.
Both have been tested under Windows XP (32-bit) and Windows 7, both 32-bit and 64-bit versions.

Obtain the executables from AVT's web site. Place them in a folder of your choosing, or on the desktop.

When needed, you launch the one you want to use by double clicking on it. They are very easy to use and do not need any explanation or instruction. However, feel free to contact AVT if you have any questions. All contact information is on the last page of this manual.

4.1.1.1 Set IP Address

This PC application is named: "IPSetup.exe"

The 'IPSetup' application will 'try' 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-423, this application will likely not be able to 'find' it. To 'fix' this, temporarily change the subnet mask and/or IP address of your client computer to something in the same domain as the AVT-423.

For example, the factory default IP address of the AVT-423 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-423. 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.

4.1.1.2 Software Update

This PC application is named: "AutoUpdate.exe"

The 'AutoUpdate' application will allow you to update the AVT-423 operation firmware.

You will need to know the IP address of the AVT-423 you want to update.

You will need the new AVT-423 operation firmware file. This file is posted on AVT's web site along with the software update application.

5. Client Computer Connection

The AVT-423 is an Ethernet TCP/IP server.
The user or test computer is, therefore, a client.

5.1 AVT-423 Connection to Client Computer

5.1.1.1 Ethernet IP Address

The factory default IP address of the AVT-423 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-423 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-423 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 also obtain the MAC address from the client computer ARP table. One way this can be done is to connect the AVT-423 to the network. From the client computer, open a command window. Ping the AVT-423 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.

Xxx mike – is there a way to query the AVT-423 for its MAC address ??

5.1.1.3 TCP/IP Port

Communications with the AVT-423 vehicle network interface is via port 10001.

All communications with the AVT-423 vehicle interface is in binary bytes [not ASCII hex]. Refer to Section 5.2 for a description of the 'packetized' communications protocol between the AVT-423 and the client computer. All communications with the AVT-423 follow the exact same rules and formats as that of the AVT-423 and all other AVT interface equipment.

5.1.2 Ethernet IP Addressing Modes

Two IP addressing modes are available for the AVT-423.

- Static
- DHCP

5.1.2.1 Static IP Addressing

The factory default addressing mode for the AVT-423 is static and the address is set to 192.168.1.70. In static mode the Ethernet address of the AVT-423 is always the same and does not change when power is cycled.

5.1.2.2 DHCP Addressing

Setting the AVT-423 IP address to 0.0.0.0 will enable DHCP (Dynamic Host Configuration Protocol) function.

In this mode, the AVT-423 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-423 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-423 you should use the Netburner supplied software; described above in Section 4.1.1.1.

5.2 Packet Communications Between the AVT-423 and the Client Computer

Communications between the client computer and the AVT-423, in both directions, uses a 'packet' protocol. This is the same protocol or method used by all AVT interface hardware.

- 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.
- 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: EDL and BRS bits are set; channel 03; CAN3.

$05:	receive object 05.
$07 77:	CAN frame ID.
01 02 03, …	data bytes.

Additional information about the AVT protocol is available at the beginning of the "Master Commands and Responses" document available from our web site at:

www.AVT-HQ.com/download.htm#Notes

6. Power and Network Connection

The power and network connector (P3) 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.

PC board revision level can be found in copper on the bottom of the PC board.

Board configuration revision level is marked in the white block on the top of the board.

6.1 AVT-423 Board Revision "B" & "C"

Pin #	Description	Notes
1	CAN0_H	
14	CAN0_L	
2	CAN1_H	Transceiver software selected.
15	CAN1_L	Transceiver software selected.
3	CAN2_H	
16	CAN2_L	
4	CAN3_H	
17	CAN3_L	
5	FR-BUS_A_P	
18	FR-BUS_A_M	
6	FR-BUS_B_P	
19	FR_BUS_B_M	
7		
20		
8		
21		
9		

22		
10	CAN1_SWC	Transceiver software selected.
23	LIN0	
11	LIN1	KWP is alternate
24	GND	Same as pin # 12
12	GND	Same as pin # 24
25	RAW_VIN	Same as pin # 13
13	RAW_VIN	Same as pin # 25

P3 (the DB-25P connector on the AVT-423 board)
Table 1

6.2 Power Requirements

The AVT-423 board requires a suitable external power supply. Fairly clean +8 to +18 VDC.

6.2.1 Ground

Common ground is required between the AVT-423 board and all connected devices. On P3 there are two 'ground' pins, #12 and #24. Both are connected directly to the ground plane of the AVT-423 board. Only one is needed for normal operations.

6.2.2 Input Voltage

The external power supply is connected to P3 pin #13 or #25. The two pins are connected together, internally, on the AVT-423 board. Only one is needed for normal operations.

6.2.3 Power Dissipation

Power dissipation of the AVT-423 is listed here.

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. Flexray was not installed.

Input Voltage	Min / Max/ Ave Measured Current	Power
+8 VDC	380 / 488 / 419 mA	3.4 W
+10 VDC	304 / 392 / 337 mA	3.4 W
+12 VDC	270 / 338 / 298 mA	3.6 W
+15 VDC	213 / 263 / 231 mA	3.5 W
+18 VDC	218 / 179 / 195 mA	3.5 W

6.2.4 Fuse

The AVT-423 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.)

The fuse will blow due to an over-current condition. It will also blow in an over-voltage condition (crowbar). The over-voltage threshold is approximately +33 (+/- 1.65) VDC.

7. ADC Connection

None at this time. Possible – future hardware upgrade to add several channels of Analog to Digital Converter capability.

8. DAC Connection

None at this time. Possible – future hardware upgrade to add one or two channels of Digital to Analog Converter capability.

9. Operation Modes

Unlike many other AVT interfaces, the AVT-423 does not have "Operation Modes".

This interface comes 'alive' with all networks initialized and disabled.

9.1 Simultaneous Network Operations

With two exceptions, all networks can be operated simultaneously.

Exception #1.
CAN1 is user selected to be either a 2-wire or a Single Wire CAN channel.
They are separate transceivers. You can not use both at the same time.

Exception #2.
KWP operations use the LIN1 transceiver. Thus, LIN1 can be operated as LIN (channel 5) or KWP (channel 6) but not both at the same time.

10. Network Hardware Descriptions

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

10.1 CAN0 - 2-wire CAN

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

It uses the Microchip MCP2562FD-E/SN transceiver.

CAN0 is not CAN-FD capable.

Refer to Table 1 for pin and signal definitions.

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

The AVT-423 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.

Other termination configurations, including Ford compliant AC termination, are available - contact the factory for details.

10.1.1 CAN0 Channel Number

For the user, CAN0 is designated channel 0.

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

10.2 CAN1 - 2-wire CAN or Single Wire CAN

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

For 2-wire operations it uses the Microchip MCP2562FD-E/SN transceiver.

For single wire operations it uses the On-Semi NCV7356D2G transceiver.

CAN1 is not CAN-FD capable.

Refer to Table 1 for pin and signal definitions.

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

The AVT-423 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.

Other termination configurations, including Ford compliant AC termination, are available - contact the factory for details.

10.2.1 CAN1 Channel Number

For the user, CAN1 is designated channel 1.

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

10.3 CAN2 - 2-wire CAN

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

It uses the Microchip MCP2562FD-E/SN transceiver.

CAN2 is both CAN (non-FD) and CAN-FD capable.

Refer to Table 1 for pin and signal definitions.

Termination can be user selected to be ‘in’ or ‘out’. The default is ‘in’.
Refer to the ‘$7x 62’ command.

The AVT-423 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.

Other termination configurations, including Ford compliant AC termination, are available - contact the factory for details.

10.3.1 CAN2 Channel Number

For the user, CAN2 is designated channel 2.

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

10.4 CAN3 - 2-wire CAN

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

It uses the Microchip MCP2562FD-E/SN transceiver.

CAN3 is both CAN (non-FD) and CAN-FD capable.

Refer to Table 1 for pin and signal definitions.

Termination can be user selected to be ‘in’ or ‘out’. The default is ‘in’.
Refer to the ‘$7x 62’ command.

The AVT-423 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.

Other termination configurations, including Ford compliant AC termination, are available - contact the factory for details.

10.4.1 CAN3 Channel Number

For the user, CAN3 is designated channel 3.

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

10.5 LIN0

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

The AVT-423 uses the NXP (Freescale) MC33660 transceiver. Maximum baud rate is inferred to be 150 kbps. A 1 Kohm resistor is used as the LIN bus (K-line) pull-up to the same supply voltage for the AVT-423 board.

10.5.1 LIN0 Channel Number

LIN0 is designated channel 7.

10.6 LIN1

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

The AVT-423 uses the NXP (Freescale) MC33660 transceiver. Maximum baud rate is inferred to be 150 kbps. A 1 Kohm resistor is used as the LIN bus (K-line) pull-up to the same supply voltage for the AVT-423 board.

10.6.1 LIN1 Channel Number

LIN1 is designated channel 5.

10.7 KWP

Key Word Protocol communications uses the LIN1 transceiver and associated K-line. Refer to Section 10.6, above, for information about the physical layer.

10.7.1 KWP Channel Number

KWP is designated channel 6.

10.8 Flexray

The Flexray bus is implemented using the NXP (Freescale) MC9S12XF512MLM device.

A dual-bus Flexray interface is implemented. The plans are for it to be available as a dual-bus or single-bus network interface.

The AVT-423 uses the ON Semiconductor NCV7383DB0R2G bus transceiver device with a common mode choke.

10.8.1 Flexray Channel Number

Flexray is designated channel 9.

11. CAN Channel Operations

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

Each CAN channel of the AVT-423 is independent of all other channels.
This applies to all channel parameters.

11.1 CAN and CAN-FD notes

CAN0 and CAN1 only support Classical CAN (non-FD) formatted frames.
This means 11 and 29-bit IDs, 0 to 8 data bytes, RTR is 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 user controls the format of the CAN frame through the EDL bit in the transmit command (bit 5 of the channel byte).

When the EDL bit is cleared (a '0'), the channel will send a (non-FD) CAN formatted frames.

When the EDL bit is set (a '1'), the channel will send a CAN-FD formatted frame.

The EDL bit is bit 5 of the object byte in a transmit command.

RTR is not supported in CAN-FD.

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

If RTR set, both EDL and BRS must cleared – else the command is in error.

"The command" – above refers to transmit commands (0x), 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 non-FD 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.

11.2 CAN-FD IP Core Notes

The CAN-FD IP core version number can be obtained by sending the 'B1 02' command.

All AVT-423 boards shipped prior to 22 April 2018 had CAN-FD IP core version '0004' installed.

Boards shipped after that date will have core version '0010' installed.

The only change between those two versions is the speed of the source clock for the CAN-FD controller.

11.2.1 Version 0004

In CAN-FD IP core version '0004' the source clock for the CAN-FD controller runs at 128.0000 MHz.

CAN timing register values, register definitions, and related specific baud rate information can be obtained on request.

11.2.2 Version 0010

In CAN-FD IP core version '0010' the source clock for the CAN-FD controller runs at 160.0000 MHz.

The CAN clock was increased to permit generation of 5 Mbaud rate to meet anticipated customer requirements.

CAN timing register values, register definitions, and related specific baud rate information can be obtained on request.

11.3 CAN Channel Operational Modes

Each CAN channel of the AVT-423 has three operating modes:

Disabled
Normal
Listen only. Only available for CAN0 and CAN1; not yet implemented.

11.3.1 Disabled

The CAN channel can not receive any messages and it can not transmit any messages.

Command: 73 11 0x 00 Status report: 83 11 0x 00.

11.3.2 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 11.3 for a discussion of object ID and Mask.

The CAN channel is enabled for receive and transmit.

Command: 73 11 0x 01 Status report: 83 11 0x 01.

11.3.3 Listen Only

This feature/function has not yet been implemented in firmware.

11.3.4 Transmit Command

The fields and bits construction of a transmit command are shown here. The transmit command is also explained in the Commands and Responses – Section 15.

There are three forms of the transmit command. The number of bytes in the transmit command determines the format of the command to 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 also be expressed as '$12 00 xx'.

11.3.4.1 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. Refer to the beginning of Section 15 for a complete description of all the transmit command formats.

11.3.4.2 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. Refer to the beginning of Section 15 for a complete description of all the transmit command formats.

11.3.4.3 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. Refer to the beginning of Section 15 for a complete description of all the transmit command formats.

11.3.4.4 CAN0 and CAN1 Byte Count Limits

The total number of data bytes permitted in a CAN transmit command depends on whether or not ISO 15765 processing is enabled for the specified transmit object.

11.3.4.5 CAN2 and CAN3 Byte Count Limits

CAN2 and CAN3 support both (non-FD) CAN and CAN-FD formatted frames.

When a transmit command specifies a (non-FD) CAN frame (EDL 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 EDL 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 this 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-423 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-423 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).
- 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).

11.3.5 Receive Response

There are two possible 'from the network' responses that the AVT-423 can send to the client.

1. A CAN message from the network (from another CAN node).

2. A transmit acknowledgement; aka: a transmit ack.

Both are described at the beginning of Section 15.1.

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-423 interface.

11.3.5.1 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. Refer to Section 15.1 for a description of all bytes in the packet.

11.3.5.2 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. Refer to Section 15.1 for a description of all bytes in the packet.

11.3.5.3 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. Refer to Section 15.1 for a description of all bytes in the packet.

11.3.6 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.

11.3.6.1 CAN0 and CAN1 Time Stamp Clock

For CAN0 and CAN1, the time stamp 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.

Also available is a 1 msec time stamp source. It is common to all other channels (if the 1 msec clock is selected). The time stamp rolls over at $FFFFFFFF.

11.3.6.2 CAN2 and CAN3 Time Stamp Clock

For CAN2 and CAN3, the time stamp is a 32-bit free running counter that is driven by a 2 kHz clock. As a result, the time stamp increment is 0.5 msec.

It appears that the time stamp clock and counter are shared for CAN2 and CAN3 (they both read the same counter.

The time stamp rolls over at $FFFFFFFF.

Also available is a 1 msec time stamp source. It is common to all other channels (if the 1 msec clock is selected). The time stamp rolls over at $FFFFFFFF.

11.3.6.3 Transmit Acknowledgment Description

Refer to Section 15.1 for a complete description of the transmit ack response to the client with and without time stamps.

11.3.6.4 Receive Message Description

Refer to Section 15.1 for a complete description of the receive message response to the client with and without time stamps.

11.4 Object ID and Mask

11.4.1 Configuration

Each CAN channel of the AVT-423 is independent of all other channels.

Each CAN channel of the AVT-423 has 16message objects numbered: $00 to $0F.

For CAN0 and CAN1 each message object can be configured as either transmit or receive.

For CAN2 and CAN3 each message object is receive only.

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.

Note that there are a few differences between CAN0 / CAN1 and CAN2 / CAN3.

11.4.2 Object ID and Mask Operation

Object IDs and Masks are associated as pairs.

Object ID0 is paired to Mask0; object ID1 is paired to Mask1, etc.

Using the ‘$75 2A …’ form of the object command specifies an 11-bit ID.

Using the ‘$77 2A …’ form of the object command specifies a 29-bit ID.

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.

11.4.3 Acceptance ID and Mask Notes

The Acceptance ID commands: '7x 05' and '7x 2A' are identical and redundant. The user can use either one, as they please.

The user should configure the Acceptance ID (commands: '7x 05' or '7x 2A') before setting the Mask (command: '7x 2C'). The order does not affect operations, but it will affect the 'look' of the mask response.

11.4.4 CAN0 and CAN1 specifics

For the '7x 18' periodic message setup command, the user must specify the IDE and RTR bits. They are bits 7 and 6 (respectively) in the channel byte of the '7x 18' command.

For the '7x 05' and '7x 2A' commands the user must specify the RTR bit. It is bit 6 of the object byte of the command. The user does not specify the IDE bit as the format of the command indicates 11 or 29-bit ID.

For the'0x' transmit command the user must specify the IDE and RTR bits. They are bits 7 and 6 (respectively) of the object byte of the transmit command.

For the '7x 2C' acceptance mask command there is no mask bit for either IDE or RTR bits.

11.4.5 CAN2 and CAN3 specifics

For the '7x 18' periodic message setup command, the user must specify the IDE, RTR, EDL, and BRS bits. They are bits 7, 6, 5, and 4 (respectively) in the channel byte of the '7x 18' command.

For the '7x 05' and '7x 2A' commands the user must specify the RTR and EDL bits. They are bits 6 and 5 (respectively) in the object byte of the command. The user does not specify the IDE bit as the format of the command indicates 11 or 29-bit ID.

For the '0x' transmit command the user must specify the IDE, RTR, EDL, and BRS bits. They are bits 7, 6, 5 and 4 (respectively) of the object byte of the command.

For the '7x 2C' acceptance mask command the user must specify mask bits for IDE and EDL. They are bits 7 and 5 (respectively) of the object byte of the command.

11.4.5.1 ID / Mask Example

Channel CAN0. Use object $04. Desired message is a 29-bit ID = 12 34 56 78, all bits are "must match". RTR bit is 0 (do not receive RTR frames).

The following commands are used.

```
; set CAN0 object # 4 ID
77 2A 00 04 12 34 56 78

; set CAN0 mask # 04
77 2C 00 04 1F FF FF FF
```

Only network messages with a 29-bit ID = 12 34 56 78 and RTR = 0 will be received.
(It is assumed the operator has completed all other necessary channel initialization commands.)

11.5 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:
   set the mode (receive or transmit),
   if receive, set the ID,
   if receive, set the mask,
   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.

11.5.1 Communications Example

Set up CAN1, use object #0 for receive, object #5 for transmit.

; set CAN1 to 500 Kbaud
73 0A 01 02

; set CAN1 ID0 = 07 E0
75 2A 01 00 07 E0

; set CAN1 mask0 low order 4-bits are don't care.
75 2C 01 00 07 F0

; enable CAN1 object #0 for receive
74 04 01 00 01

; [optional] enable CAN1 object #5 for transmit
74 04 01 05 02

; enable CAN1 for normal operations
73 11 01 01

; send a message with 5 bytes in the data field to ID = 07 80 using object #5.
09 01 05 07 80 04 11 22 33 44

; receive the transmit ack = 02 01 A5

; receive a message from the network = 0C 01 00 07 E3 05 AA BB CC DD EE 00 00

11.6 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 rate.
4. Set up the object(s) you want to use.
5. For each receive object:
   set the ID,
   set the mask,
   enable the object.
6. Enable the CAN channel.

11.6.1 Classical CAN Communications Example

Set up CAN3, use object #A receive 11-bit ID messages of the form 07 Ex, transmit ID = 07 80, send one message, receive one message.

```
; set CAN3 to 1 Mbaud
73 0A 03 01

; set CAN3 IDA = 07 E0
75 2A 03 0A 07 E0

; set CAN3 maskA low order 4-bits are don't care.
75 2C 03 0A 07 F0

; enable object $A for receive
74 04 03 0A 01

; enable CAN3 for normal operations
73 11 03 01

; send a message with 5 bytes in the data field to ID = 07 80
09 03 00 07 80 04 11 22 33 44

; receive the transmit ack  =  02 03 A0

; receive a message from the network  =  0C 03 0A 07 E3 05 AA BB CC DD EE 00 00
```

11.6.2 CAN-FD Communications Example

Set up CAN2, use object $C receive 29-bit ID messages with ID = 12 34 56 78, CAN-FD format, transmit ID = 12 AB CD EE, CAN-FD format, data field at high speed, send one message, receive one message.

```
; set CAN2 to 500 Kbaud and 2 Mbaud
74 0A 02 02 0C

; set CAN2 IDC = 12 34 56 78
77 2A 02 BC 12 34 56 78

; set CAN2 maskC to all must match.
77 2C 02 BC 1F FF FF FF
```

; enable object $C for receive
74 04 02 0C 01

; enable CAN2 for normal operations
73 11 02 01

; send a message with 12 bytes in the data field to ID = 12 AB CD EE
11 12 02 B0 12 AB CD EE 01 02 03 04 05 06 07 08 09 0A 0B 0C

; receive the transmit ack = 02 02 A0

; receive a message from the network = 11 12 02 BC 12 34 56 78 0A 0B 0C 0D 0E 0F 10 11 12 13 14 15

11.6.3 Transmit Attempt Limits

The CAN protocol requires that there be at least two functioning nodes on a CAN bus. When one node transmits a CAN frame it will expect at least one other node to assert an ACK bit at the very end of the frame. A transmitting node can not ack it's own transmissions.

If a CAN node does not see the ACK bit, it will immediately transmit the frame again. For many CAN controllers this behavior will continue indefinitely until the CAN controller is commanded to abort or is reset.

In the AVT-423, channels CAN0 and CAN1 will act this way.

However, channels CAN2 and CAN3 will limit the number of transmit attempts before automatically terminating the transmissions and deleting that frame. The number of attempts defaults to 400 (0x0190) for each channel.

The '7x 63' command allows the user to set the number of attempts. A value of '0x0000' means try indefinitely (no limit).

Note 1: This behavior is still linked to the error counters. For example, if the limit is left set at 400 attempts an error response will still be generated and sent to the client when the error counter passes 127. Similarly, if the number of attempts is set below that threshold, no error response will be generated – even though the transmit attempt failed.

Note 2: The AVT-423 will send a '02 0x A0' transmit ack to the client when the number of transmit attempts limit is reached and the message is discarded. The object number will be zero if the message is discarded, regardless of the object number specified in the transmit command.

11.7 Periodic Message Support

At this time only Type1 periodic messages are supported. Type1 operations is the default for all periodic messages.

11.7.1 Type1 Periodic Messages

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 (ID and data field are defined).
The interval count is defined.
The message is enabled.

11.7.1.1 Type1 Example

Want to send two messages on CAN2 at 500 Kbaud and (non-FD) CAN format. One message every 500 msec. The other message every 750 msec. Using CAN2 Type1 operations, here is a sequence of commands to do this.

1. ; Set CAN2 baud rate to 500 kbps
   73 0A 02 02
2. ; Enable CAN2 for normal operations
   73 11 02 01
3. ; Define periodic message $01, ID = 246, data = 03 A3 B4 C5
   79 18 02 01 02 46 03 A3 B4 C5
4. ; Set periodic message $01 for an interval count of 1000 = 1000 msec.
   75 1B 02 01 03 E8
5. ; Enable periodic message $01
   74 1A 02 01 01
6. ; CAN2, periodic message $01 is now active.
7. ; Define periodic message $06, ID = 498, data = 04 1A 2B 3C 4D
   7A 18 02 06 04 98 04 1A 2B 3C 4D
8. ; Set periodic message $06 for an interval count of 500 = 500 msec.
   75 1B 02 06 01 F4.
9. ; Enable periodic message $06
   74 1A 02 06 01
10. CAN2, periodic message $06 is now active.

11.7.2 Type2 Periodic Messages

This feature/function has not yet been implemented in firmware.

Any periodic messages that are designated as Type2 are transmitted in sequence (not independently).

If any messages for a CAN channel are to be set up for Type2 operations, the first periodic message must be used and designated for Type2. The timer of the first periodic message is used as the timer for all Type2 messages on that channel.

For example, set-up periodic message $00 (the first message) ID and data field. Set PM $00 timer increment for 100 msec. ($64). Set-up PM $05 and $18 ID and data. Their timers are not used. Each time the timer for PM $00 expires, the next periodic message will be queued for transmission. Thus, PM $00, $05 $18 will transmitted in sequence. At the end of the sequence, the function loops back to the top.

11.7.3 Periodic Message Commands

All commands are listed in Section 15. A brief summary is provided here.

- 7x 18 Define a periodic message.
- 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.

11.8 Periodic Message Special Functions

These features/functions have not yet been implemented in firmware.

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.

Only one mode is allowed to be enabled for any given periodic message. If you attempt to enable more than one mode, the last mode command will be the one 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.

11.8.1 CAN Frame Data Definition

For each periodic message, the CAN frame can contain up to 8 data bytes.

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 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).

11.8.2 Special Function xxx

xxx

11.8.3 Special Function xxx

xxx

11.8.4 Special Function xxx

xxx

11.9 Periodic Pause Function

This function has not yet been implemented in firmware.

The Periodic Pause function, when enabled for a specific CAN channel, will inhibit all CAN periodic messages whenever an ISO 15765 transaction is in-progress. Note that this only applies to ISO 15765 transactions that require more than one CAN frame; in other words, if a multi-frame message transaction is in-progress on the CAN channel, no periodic messages will be queued for transmission.

11.9.1.1 Periodic Pause Function Command

This function has not yet been implemented in firmware.

Refer to the $7x 1F command in Section 15 for detailed information regarding the command format.

11.10 ISO 15765 Support for CAN0 and CAN1

This function and the related commands are new as of firmware version 0012.

This Section is for channels CAN0 and CAN1 only.

The ISO 15765 capability for the AVT-423 was based on the AVT-718.
(For those users who are familiar with that model.)

Refer to Section 11.11 for information about ISO 15765 support for channels CAN2 and CAN3.

11.10.1 Basic Set-up and Operational Discussion

For one channel, the user specifies:

A transmit object.
A receive object.

The user sets-up each object by specifying:

The message ID (11 or 29-bit)
The direction: transmit or receive.

The user 'pairs' those two objects which enables ISO 15765 processing for those two objects.

The user specifies if the 'AE' (address extension) is to be used or not.

The user specifies if message padding is to be used or not.

Any CAN frames received through the designated ISO 15765 receive object is 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 they should only include the message ID (11 or 29-bit), the 'AE' byte (if used), and the data bytes.

The user must omit the PCI byte and all pad bytes from the transmit command.

The AVT-423 will process, format, transmit and handle all ISO 15765 handshaking as required. The client is not involved in any of that.

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 user 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 me.
I'll be glad to help.

A set-up and operational example is the best way to demonstrate this ISO 15765 capability.

Remember: The AVT-423 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.

11.10.2 Differences with ISO 15765 on CAN0/1 (as compared to CAN2/3)

Channels CAN0 and CAN1 have a total of 16 objects (numbered $0 to $F).
An object can be configured for receive or transmit. Only one configuration per object.

Both transmit and receive objects are configured using the '7x 05' command and enabled using the '7x 04' command.

11.10.3 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.

11.10.3.1 ISO 15765 Initialization Example

The following command sequence will initialize the AVT-423 using the scenario described above.

At each step the description is first, then the command. All commands are Hex digits. No "$" or "0x" prefixes are used here.

Refer to Section 16 for complete and detailed description of each command.

- Set CAN0 to 500 Kbaud.
  74 0A 00 02
- Object #2 ID = $246.
  75 05 00 02 02 46

- Enable object #2 for transmit.
  74 04 00 02 02
- Object #3 ID = 0357.
  75 05 00 03 03 57
- Object #3 acceptance mask, all bits must match.
  75 2C 00 03 07 FF
- Enable object #3 for receive.
  74 04 00 03 01
- Pair objects #2 and #3. 'AE' is disabled.
  74 28 00 02 03
  (Order of the objects is not important.)
- Enable padding for the transmit object. Pad byte = FF.
  75 27 00 02 01 FF
- Enable CAN0 for operations.
  73 11 00 01

At this point all communications through these two objects are handled as ISO 15765 formatted messages.

11.10.3.2 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).

11.10.3.3 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. 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 user 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 user does not know that three CAN frames were transmitted and other handshaking conducted between the AVT-423 and the module to obtain this complete response.

11.11 ISO 15765 Support for CAN2 and CAN3

This function and the related commands are new as of firmware version 0020.

This Section is for channels CAN2 and CAN3 only.

These channels support ISO 15765 operations for both Classical CAN as well as CAN-FD.

Refer to Section 11.10 for information about ISO 15765 support for channels CAN0 and CAN1.

11.11.1 Basic Set-up and Operational Discussion

For one channel, the user specifies:

A transmit object.
A receive object.

For channels CAN2 and CAN3, there are 16 receive objects (numbered $0 to $F) and 16 transmit objects (numbered $0 to $F).
(This is very different than channels CAN0 and CAN1.)

The user sets-up each object by specifying:

The message ID (11 or 29-bit),
as well as both EDL (extended data length) and BRS (baud rate switch).
The receive object is then enabled.

The user ‘pairs’ those two objects which enables ISO 15765 processing for those two objects.

The user specifies if the 'AE' (address extension) is to be used or not.

The user specifies if message padding is to be used or not.

In the case of an ISO 15675 transaction using CAN-FD – the user can also specify the maximum data count for each frame. This is known as 'max_dlc'. The default is 64 bytes of data, maximum, per frame. The '7x 29' command allows the user 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.

Any CAN frames received through the designated ISO 15765 receive object is 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 they should only include the message ID (11 or 29-bit), the 'AE' byte (if used), and the data bytes.

The user must omit the PCI byte and all pad bytes from the transmit command.

The AVT-423 will process, format, transmit and handle all ISO 15765 handshaking as required.
The client is not involved in any of that.

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 user 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 me.
I'll be glad to help.

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-423 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.

11.11.2 Differences with ISO 15765 on CAN2/3 (as compared to CAN0/1)

Channels CAN2 and CAN3 have 16 transmit objects (numbered $0 to $F).
There are also 16 receive objects (numbered $0 to $F).

Transmit objects are configured using the '7x 17' command and do not have a separate enable command.

Receive objects are configured using the '7x 05' command and enabled using the '7x 04' command.

11.11.3 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.

11.11.3.1 ISO 15765 Initialization Example – Classical CAN

The following command sequence will initialize the AVT-423 using the scenario described above.

At each step the description is first, then the command. All commands are Hex digits. No "$" or "0x" prefixes are used here.

Refer to Section 16 for complete and detailed description of each command.

- Set CAN2 to 500 Kbaud.
  74 0A 02 02
  (Classical CAN will be used, so the 'fast' baud rate can be omitted from the command.)
- Object #4, transmit, ID = $357.
  75 17 02 04 03 57
- Object #8, receive, ID = 0246.
  75 05 02 08 02 46
- Object #8 acceptance mask, all bits must match.
  75 2C 02 38 07 FF
  (EDL and BRS bits are set for 'must match'.)
- Enable object #8 for receive.
  74 04 02 08 01
- Pair objects #4 and #8. 'AE' is disabled.
  74 28 02 04 08
  (Order of the objects is not important.)
- Enable padding for the transmit object. Pad byte = FF.
  75 27 02 04 01 FF
- Enable CAN2 for operations.
  73 11 02 01

At this point all communications through these two objects are handled as ISO 15765 formatted messages.

11.11.3.2 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).

11.11.3.3 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. 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 user 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, EDL, and BRS bits are all 0.

02 46: received message ID.

01 02 … : the data the module sent.

Note that the user does not know that three CAN frames were transmitted and other handshaking conducted between the AVT-423 and the module to obtain this complete response.

11.11.4 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.

11.11.4.1 ISO 15765 Initialization Example –CAN-FD

The following command sequence will initialize the AVT-423 using the scenario described above.

At each step the description is first, then the command. All commands are Hex digits. No "$" or "0x" prefixes are used here.

Refer to Section 16 for complete and detailed description of each command.

- Set CAN2 to 500 Kbaud and 2 Mbaud.
  75 0A 02 02 0C
- Object #4, transmit, ID = $357.
  75 17 02 34 03 57
  (EDL and BRS bits in the object byte are set to indicate CAN-FD.)
- Object #8, receive, ID = $246.
  75 05 02 08 02 46
- Object #8 acceptance mask, all bits must match.
  75 2C 02 38 07 FF
  (EDL and BRS bits in the object byte are set, which is 'must match'.)
- Enable object #8 for receive.
  74 04 02 08 01
- If necessary, specify the maximum number of bytes in the data field.
  (Use the '7x 29' command.)
- Pair objects #4 and #8. 'AE' is disabled.
  74 28 02 04 08
  (Order of the objects is not important.)
- Enable padding for the transmit object. Pad byte = FF.
  75 27 02 04 01 FF

- Enable CAN2 for operations.
  73 11 02 01

At this point all communications through these two objects are handled as ISO 15765 formatted messages.

11.11.4.2 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).
EDL 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).

11.11.4.3 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.

The response the user 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; EDL = 1; BRS = 1.

02 46: received message ID.

01 02 … : the data the module sent.

12. LIN1 Operations

LIN1 operation is independent of all other channels.

LIN1 operation is controlled by the $53 69 0x 0y command.

LIN1 supports LIN revisions: 1.2, 1.3, 2.0, 2.1, 2.2A.

LIN1 hardware is shared with KWP operations. Only one mode can be enabled at a time. Either LIN1 or KWP, but not both.

12.1 Communications

When enabled, LIN1 will passively receive all messages from the LIN1 bus.

The AVT-423 is capable of transmitting to the LIN1 bus as a Master without data, as a Master with data, or as a Slave with data.

12.1.1 Message Length

LIN protocol specification revision 2.0 and later eliminated the relationship between message ID and expected frame length. Current AVT-423 firmware does not provide so-called “ID byte processing” of received messages.

To determine the end of a LIN frame, the AVT-423 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 user 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.

12.1.2 Checksum

Both Classic and Enhanced checksum methods are available through the $53 5A command.

This 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.

12.1.3 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-85x will throw out an ID byte only message and not tell the client.

The $53 66 command selects whether or not the AVT-85x informs the client that an ID byte only message was received.

The $53 66 01 01 command causes the AVT-423 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.
```

12.1.4 Communications Example

This example is enable LIN1 operation, receive a message from the LIN1 bus (passively) and to send a message to the LIN1 bus using the three possible transmit formats. Time stamps are disabled.

```
; enable LIN1 operations
53 69 05 01

; receive a LIN network message  (passively)
05 05 00 C4 78 9A
;       0  indicates from the network
;       5  count of bytes to follow
;       05  channel 5 - LIN
;       00  status byte, no bits set indicates no errors detected
;       C4  message ID
;       78 9A  message data field

;  send message as a Master without data  --  this elicits a response from a Slave node
03 05 01 25
;       0  indicates to the network
;       3  count of bytes to follow
;       05  channel 5 - LIN
;       01  master node
;       25  message ID

;  send message as a Master with data  --  this sends a complete message onto the network
0B 05 01 B4 11 22 33 44 55 66 77 88
;       0  indicates to the network
;       B  count of bytes to follow = $B = 11 decimal
;       05  channel 5 - LIN
```

```
;       01  master node
;       B4  message ID
;       11 22 33 44 55 66 77 88  message data

; act as a Slave  --  the node will wait for the Master to request data from the specified ID
05 05 00 C4 11 22
;       0  indicates to the network
;       5  count of bytes to follow
;       05  channel 5 - LIN
;       00  slave node
;       C4  message ID
;       11 22  message data
```

12.1.5 Time Stamp

Time stamps for both the transmit ack and received messages can be disabled or enabled using the 5x 08 command.

Time stampss are four-bytes (32-bits) taken from a free running 1 msec timer. The time stamp rolls over at $FFFFFFFF.

Transmit ack: the time stamp is a four-byte value immediately after the packet header byte; but before the LIN channel number (05).

Received message: the time stamp is a four-byte value immediately after the packet header byte; but before the LIN channel number (05).

12.1.5.1 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 - LIN
        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 - LIN
        00  status byte indicating no errors detected.
        25  message ID.
        11 22 33 44  message bytes.
```

12.1.5.2 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 - LIN
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 - LIN
40 status byte, bit 5 set, indicates from this node.

12.2 Special Functions

At present, there are no special functions defined for LIN1 operations.

12.2.1 LIN Frame Data Definition

Each LIN frame can contain up to 8 data bytes.

In the following discussion, 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).

12.2.2 Special Function 0

Does not currently exist.

12.3 Periodic Message Support

When LIN1 is enabled, the AVT-423 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-423 unit will then transmit those messages, at the defined interval, without any operator intervention.

The AVT-423 will not generate a transmit ack when a periodic message is transmitted, unless the transmit acknowledgement function is enabled ($5x 40 command).

12.3.1 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.

Type2 periodic message are not implemented.

A periodic message designated as Slave operate as described in Section 12.3.4, below.

12.3.2 Organization of Periodic Messages

There are sixteen (decimal) periodic messages. The periodic messages are numbered $00 to $0F (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.

12.3.3 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.

12.3.3.1 Type1 Example

Here is a sequence of commands to set-up and enable a Master periodic message. Note: LIN1 is channel 05.

1. ; Enable LIN1 operations.
   53 69 05 01
2. ; Define LIN1 periodic message $02.
   ; the message is: Master, ID = 25, data = 68 6A F1 3F
   79 18 05 02 01 25 68 6A F1 3F
3. ; Set LIN1 periodic message $02 for an interval count of 500 = 500 msec.
   75 1B 05 02 01 F4
4. ; Enable LIN1 periodic message $02
   74 1A 05 01 01
5. ; LIN1 Periodic message $02 will begin transmitting.

12.3.4 Slave Periodic Message

This only applies to periodic messages that are designated as slave when the periodic message 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 LIN1 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 LIN1 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.

12.3.5 Periodic Message Commands

All commands are listed in Section 15 with full definition. A brief summary is provided here.

- $7x 18 Define a periodic message.

- $7x 1B Periodic message interval.
- $7x 1A Periodic message disable/enable.
- $7x 1C Disable all periodic messages.

12.4 Periodic Message Special Functions

At present, there are no special functions defined for LIN1 periodic messages.

12.4.1 LIN Frame Data Definition

Each LIN frame can contain up to 8 data bytes.

In the following discussion, 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).

12.4.2 Periodic Message Special Function 0

Does not currently exist.

12.4.2.1 PM-SF0 Command

Does not currently exist.

12.5 ABIC Support

LIN1 does NOT support communications with an ABIC module.

12.6 Commands and Responses

Refer to Section 15 for a complete list of LIN1 commands and Section 15.1 for responses.

13. LIN0 operations

LIN0 operation is independent of all other channels.

LIN0 operation is controlled by the $53 69 0y 0z command.

LIN0 supports LIN revisions: 1.2, 1.3, 2.0, 2.1, 2.2A.

LIN0 does NOT have an alternate function.

13.1 LIN0 Operation Notes

LIN0 operations are nearly identical to LIN1 described in Section 12, above.

LIN0 is channel 7.

13.2 Commands and Responses

Refer to Section 15 for a complete list of LIN0 commands and Section 15.1 for responses.

14. KWP operations – using LIN1 Hardware

This function has not yet been implemented in firmware.

15. Commands

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

All transmit command forms are equal in ascending order.

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

0: Packet for transmission to the network.

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 non-FD frame).
    - 0: normal frame.
    - 1: RTR true, remote transmit request.
  - b5: EDL.
    - 0: non-FD 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

0x 0z 0m id aa bb cc … :

x: count of bytes to follow.
z: channel: 7, 5.
m: slave / master.
0: slave.
1: master.
id: LIN message ID. (User must include parity bits.)
aa bb cc … data [optional].

1: CAN packet for transmission to the network; alternate header formats.

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 non-FD frame).
0: normal frame.
1: RTR true, remote transmit request.
b5: EDL.
0: non-FD 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 non-FD frame).
0: normal frame.
1: RTR true, remote transmit request.
b5: EDL.
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.

Data byte count limitations CAN2 and CAN3

Maximum is 64 bytes with EDL bit = 1 and non-ISO 15765 operations.
See Section 11.2.3.5 for list of valid data field sizes.
Maximum is $2000 (8192 decimal) for ISO 15765 enabled object.

2: Reset.

21 10: Reset CAN0.
21 11: Reset CAN1.
21 12: Reset CAN2.
21 13: Reset CAN3.
21 20: Reset LIN0.
21 21: Reset LIN1.
21 30: Reset Flexray hardware (hard reset).

3:

4:

5: Configuration.

LIN0, LIN1

52 01 0r Send received checksum to client, status query.
r: channel: 7, 5.

53 01 0r 0z Send received checksum to client.
r: channel: 7, 5.
z: 0: disabled. [Default.]
1: enabled.

LIN0, LIN1

52 02 0r Receive buffer timeout query.
r: channel: 7, 5.

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

CAN0, CAN1, CAN2, CAN3, LIN0, LIN1, KWP

52 08 0r Time stamp status query.
r: channel: 0, 1, 2, 3, 7, 5, 6.

53 08 0x 0y: Disable / Enable time stamp.
r: channel: 0, 1, 2, 3, 5, 6, 7.
y: 0: Disable. [Default.]
1: Enable – uses 1 msec timer.
2: Enable – uses native timer.
(CAN0 and CAN1: baud clock.)
(CAN2 and CAN3: 2 kHz clock.)

LIN0, LIN1

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

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

CAN0, CAN1, CAN2, CAN3, LIN0, LIN1, KWP

52 40 0r Transmit ack status query.
r: channel: 0, 1, 2, 3, 7, 5, 6.

53 40 0r 0y: Disable / Enable transmit acks.
r: channel: 0, 1, 2, 3, 7, 5, 6.
y: 0 disable.
1 enable. [Default.]

LIN0, LIN1

52 50 0r Baud rate query.
r: channel: 7, 5.

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

54 50 0r yy zz Set baud rate.
r: channel: 7, 5.

yy zz: divisor load.
125,000,000 / 32 / yy zz
(where 'yy zz' is converted to decimal)

LIN0, LIN1

52 52 0r Maximum frame time query.
r: channel: 7, 5.

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

LIN0, LIN1

52 5A 0r Checksum type query.
r: channel: 7, 5.

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

LIN0, LIN1

52 66 0r "ID byte only" error message to client, status query.
r: channel: 7, 5.

53 66 0r 0z Send "ID byte only" error message to client.
r: channel: 7, 5.
z: 0: disabled. [Default.]
1: enabled.

LIN0, LIN1

52 69 0r Secondary operations status query.
r: channel: 7, 5.

53 69 0r 0z Set secondary operations.
r: channel: 7, 5.
z: 0: disabled. [Default.]
1: enabled.

---

51 6A: Query for CPU heart beat LED blink rate.

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

LIN0, LIN1

52 7E 0r Short to ground counter reset value query.
r: channel: 7, 5.

54 7E 0r yy zz Short to ground counter reset value.
r: channel: 7, 5.
yy zz: counter reset value.
[Default = $0100 => 256.]

6: Initialization

7: CAN configuration.

CAN0, CAN1, CAN2, CAN3

72 04 0r: Object status query.
r: channel: 0, 1, 2, 3.

74 04 0r 0y 0z Disable / Enable object.
r: channel: 0, 1, 2, 3.
y: object number 0 to F.
z: 0: object disabled. [Default.]
1: object enabled for receive.
2: object enabled for transmit. (CAN0/1 only.)

CAN0, CAN1

73 05 0r 0z: Report object configuration.
r: channel: 0, 1.
z: object number: $0 to $F.

75 05 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 05 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 (Receive object only.)

73 05 0r 0z: Report object configuration.
r: channel: 2, 3.
z: object number: $0 to $F.

75 05 0r yz ss tt: Set object 11-bit ID.
r: channel: 2, 3.
y: b7: 0.
b6: 0.
b5: EDL.
0: Classical CAN frame.
1: CAN-FD frame.
b4: 0.
z: object number: $0 to $F.
ss tt: 11-bit ID, right justified.

77 05 0r yz ss tt vv ww: Set object 29-bit ID.
r: channel: 2, 3.
y: b7: 0.
b6: 0.
b5: EDL.
0: Classical CAN frame.
1: CAN-FD frame.
b4: 0.
z: object number: $0 to $F.
ss tt vv ww: 29-bit ID, right justified.

CAN0, CAN1

73 06 0r 0z: Query for object data.
r: channel: 0, 1, 2, 3.
z: object number: $0 to $F.

73 06 0r 8z: Set object for 'no' data.
- r: channel: 0, 1, 2, 3.
- z: object number: $0 to $F.

7x 06 0r 0z rr ss tt …: Set object data.
- r: channel: 0, 1, 2, 3.
- z: object number: $0 to $F.
- rr ss tt …: data

CAN0, CAN1

73 07 0r 0z: Query for object transmit status.
- r: channel: 0, 1, 2, 3.
- z: object number: $0 to $F.

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

CAN0, CAN1, CAN2, CAN3

72 08 0r: Query for transmit acknowledgement status.
- r: channel: 0, 1, 2, 3.

73 08 0r 0w: Disable / enable transmit acknowledgements.
- r: channel: 0, 1, 2, 3.
- w: 0: disable transmit acks.
  1: enable transmit acks.

CAN0, CAN1, CAN2, CAN3

Channels 1 and 2: 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: User 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:	User 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:	User 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, CAN2, CAN3

72 0B 0r: Query for Bit Timing Registers (BTR).

r:	channel: 0, 1, 2, 3.

76 0B 0r ss tt vv ww: Set Bit Timing Registers (BTR).

r:	channel: 0, 1, 2, 3.
ss tt:	Bit Timing Register 0.
vv ww:	Bit Timing Register 1.

Bit timing register values for CAN0 and CAN1: Registers 0 and 1 form a 32-bit value (where register 0 is bits 31:16 and register 1 is bits 15:0).
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 the factory for a complete definition of the bit timing registers for CAN2/3.

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 operation status.
- r: channel: 0, 1, 2, 3.

73 11 0r 0z: Set operation state.
- r: channel: 0, 1, 2, 3.
- z: 0: disabled. [Default for all CAN channels.]
  1: 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.]

CAN2, CAN3 (Transmit object only.)

73 17 0r 0z Query for transmit object configuration.
- r: CAN channel: 2, 3.
- z: 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 = is not an FD frame (EDL bit = 0)
  1 = is an FD frame (EDL bit = 1)
  b4: 0 = data at regular speed (BRS bit = 0)

1 = data at high speed (BRS bit = 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 = is not an FD frame (EDL bit = 0)
1 = is an FD frame (EDL bit = 1)
b4: 0 = data at regular speed (BRS bit = 0)
1 = data at high speed (BRS bit = 1)
z: object number: $0 to $F.
tt vv mm nn: 29-bit ID, right justified.

NOTE: The periodic message set-up command (7x 18) format is different from the AVT-853 command set. Channel and message numbers are swapped.

CAN0, CAN1

73 18 0r pp: Periodic message set-up query.
r: channel: 0, 1.
pp: message number, $00 to $2F.

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 $2F.
tt vv: 11-bit ID, right justified.
tt vv ww zz: 29-bit ID, right justified.
mm nn ...: data field.

CAN2, CAN3

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

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 non-FD frame).
0: normal frame.
1: RTR true, remote transmit request.
b5: EDL.
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 $2F.
tt vv: 11-bit ID, right justified.
tt vv ww zz: 29-bit ID, right justified.
mm nn ...: data field.

LIN0, LIN1

73 18 0r pp: Periodic message set-up query.
r: channel: 7, 5.
pp: message number, $00 to $0F.

7x 18 0r pp 0m rr ss tt … Periodic message set-up.
r: channel: 7, 5.
pp: message number, $00 to $0F.
m: 0: slave.
1: master.
rr ss tt ...: data field. [optional]

KWP

73 18 06 pp: Periodic message set-up query.
channel: 6.
pp: message number, $00 to $0F.

7x 18 06 pp rr ss tt … Periodic message set-up.
channel: 6.
pp: message number, $00 to $0F.
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 $2F.

74 19 0r pp 0y Assign object to CAN periodic message.
r: Channel: 0, 1.

pp: Message number, $00 to $2F.
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 $2F.

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

LIN0, LIN1, KWP

73 1A 0r zz: Periodic message disable/enable status query.
r: channel: 7, 5, 6.
zz: message number, $00 to $0F.

74 1A 0r zz 0v: Periodic message disable/enable.
r: channel: 7, 5, 6.
zz: message number, $00 to $0F.
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 $2F.

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

LIN0, LIN1, KWP

73 1B 0r zz: Periodic message interval count status query.
r: channel: 7, 5, 6.
zz: message number, $00 to $0F.

75 1B 0r zz vv ww: Periodic message interval count.
r: channel: 7, 5, 6.
zz: message number, $00 to $0F.
ww vv: time in milliseconds.

[Default = $03E8 => 1000 msec.]

CAN0, CAN1, CAN2, CAN3, LIN0, LIN1, KWP

72 1C 0r Disable all periodic message for channel 'r'.
- r: channel: 0, 1, 2, 3, 7, 5, 6.
- r: 'F' – disable all messages, all channels.

CAN0, CAN1, CAN2, CAN3

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.]

CAN0, CAN1, 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: $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

73 29 0r 0y: Query for 'max_dlc'.
- r: channel: 2, 3.
- y: object number: $0 to $F.

74 29 0r 0y ss: Set '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

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 yz ss tt: Set object 11-bit ID.
- r: channel: 2, 3.
- y: b7: 0.
  - b6: 0.
  - b5: EDL.
    - 0: Classical CAN frame.
    - 1: CAN-FD frame.
  - b4: 0.
- z: object number: $0 to $F.
- ss tt: 11-bit ID, right justified.

77 2A 0r yz ss tt vv ww: Set object 29-bit ID.
- r: channel: 2, 3.
- y: b7: 0.
  - b6: 0.
  - b5: EDL.
    - 0: Classical CAN frame.
    - 1: CAN-FD frame.
  - b4: 0.
- 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 yz ss tt: Specify 11-bit mask.
- r: channel: 2, 3.
- x: b7: IDE bit.
  b6: 0.
  b5: EDL bit.
  b4: 0.
- z: mask number, $0 to $F.
- ss tt: mask value, 11-bit.
  [Default = $7FF.]

77 2C 0r yz ss tt vv ww: Specify 29-bit mask.
- r: channel: 2, 3.
- x: b7: IDE bit.
  b6: 0.
  b5: EDL bit.
  b4: 0.
- 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.

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.]

CAN2, CAN3

72 63 0r: Query for transmit attempt limit.
r: channel: 2, 3.

74 63 0r ss tt: Set transmit attempt limit.
r: channel: 2, 3.
ss tt: number of attempts before message is discarded.
a value of 00 00 means indefinite (continuous, no limit).
default: 01 90 (decimal 400).

8:

9: Flexray commands.

91 30: Re-initialize all variables. NOT the same as ’21 30’ hard reset command.

91 40: Request transmit ack status.
92 40 00: Disable transmit acks.
92 40 01: Enable transmit acks.

91 6A: Flexray LED blink rate status query.
92 6A yy: Set Flexray LED blink rate.
yy: On and Off time in 10 msec increments.
00 – LED always off.
FF – LED always on.

91 B0: Flexray firmware version request.

91 D0: Flexray operational mode request.

94 D1 aa bb cc:
94 D2 aa bb cc:
95 D3 aa bb cc dd:
94 D4 aa bb cc:

91 F0: Flexray model number request.

92 F1 A5: Flexray self induced reset.

A:

B: Firmware version.
B0: Request firmware version number. (Same as B1 01.)
B1 01: Request firmware version number. (Same as B0.)
B1 02: Request FPGA version number.

C:

D: Reserved.
debug commands. do not use.

E: xxx.

F: Model Query and Reset
F0: Query for model number.

F1 A5:	Restart the AVT-423 (a form of software reset).
F1 C3:	Reset the Netburner CPU. (This will result in an Ethernet disconnect.)

15.1 Responses

High nibble, shown in left column, bits b7 - b4 indicates the Response type.
Low nibble, bits b3 – b0 indicates how many bytes are to follow.

0: Transmit acknowledgements (if enabled).

CAN0, CAN1

02 0r Az: Transmit ack.
- r: channel number: 0, 1.
- z: transmit object number.

CAN2, CAN3

02 0r A0: Transmit ack.
- r: channel number: 2, 3.

CAN0, CAN1

06 00 00 jj kk 0r Az: Transmit ack.
- 00 00 jj kk: time stamp
- r: channel number: 0, 1.
- z: transmit object number.

CAN2, CAN3

06 jj kk ll mm 0r A0: Transmit ack.
- jj kk ll mm: time stamp
- r: channel number: 2, 3.

LIN0, LIN1, KWP

02 0r pp: Transmit ack.
- r: channel number: 7, 5, 6.
- pp: receive status byte (defined below).

LIN0, LIN1, KWP

06 jj kk ll mm 0r pp: Transmit ack.
- jj kk ll mm: time stamp
- r: channel number: 7, 5, 6.
- pp: receive status byte (defined below).

0: Message received from the network.

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: EDL.
0: non-FD 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

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: 7, 5.
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: 0.
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.

1: CAN packet received from the network; alternate header formats.

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:	EDL.	
		0:	non-FD CAN frame.
		1:	CAN-FD frame.
	b4:	BRS.	
		0:	data field at normal speed.
		1:	data field at high speed.
s:	object number.		
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:	EDL.	
		0:	non-FD CAN frame.
		1:	CAN-FD frame.
	b4:	BRS.	
		0:	data field at normal speed.
		1:	data field at high speed.
s:	object number.		
tt vv:	11-bit ID, right justified.		
tt vv ww zz:	29-bit ID, right justified.		
nn pp ...:	data.		

2: Error Responses.

---

21 01 Inbound command too long, flushed.

---

21 02 FIFO2 too full, flushed and reset.

---

21 03 FIFO1 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.

---

21 06 Flexray header byte and 'new' header byte counts do not agree.

---

21 07 FIFO2 overflow, FIFO was cleared and reset.
This error response is preceded by fifteen $E0 bytes.
(The $E0 bytes are to help flush the user packet processing routine.)

---

22 08 yy Error in ’73 0A’ command. Core version number is not ‘0004’ or ‘0010’.
yy = version number read.

---

22 09 yy Error in ’74 0A’ command. Core version number is not ‘0004’ or ‘0010’.
yy = version number read.

---

22 0A yy Error in CAN2 init routine. Core version number is not ‘0004’ or ‘0010’.
yy = version number read.

---

22 0B yy Error in CAN3 init routine. Core version number is not ‘0004’ or ‘0010’.
yy = version number read.

---

22 0C yy Error in “read_core_ver” routine. Core version number is not ‘0004’ or ‘0010’.
yy = version number read.

---

21 10 CAN0 initialization error.

---

21 11 CAN1 initialization error.

---

21 12 CAN2 initialization error.

---

21 13 CAN3 initialization error.

---

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

---

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

---

22 20 yy Command processing time out.
yy: header of offending command.

---

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

---

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

---

22 41 0x Channel index error, channel 'x'.

---

22 42 0x transmit index error, xmt_ix 'x'.

---

22 43 0x buffer index error, buff_ix 'x'.

---

22 45 0x LINx transmit command too short.

---

22 46 0x LINx transmit command too long.

---

23 47 0x yy LIN transmit buffer watchdog expired.
x: channel number.
yy: buffer state.

---

23 48 0x yy LIN transmit buffer invalid state.
x: channel number.
yy: buffer state.

---

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.

---

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

(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 = 0x11, separation timer not expired, watchdog timeout.

---

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

---

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

---

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.

---
22 5F 43 canB_iso_rcv_mgr, DLC > 8 and EDL not set.

---
22 5F 44 canB_iso_rcv_mgr, DLC too short, with ae.

---
22 5F 45 canB_iso_rcv_mgr, DLC too short, without ae.

---
22 5F 46 canB_iso_rcv_mgr, (frame_cnt < pci_cnt).

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

---
22 5F 48 canB_iso_rcv_mgr, invalid buffer state.

---
22 5F 49 canB_iso_rcv_mgr, invalid consecutive frame sequence number.

---
22 5F 4A canB_iso_rcv_mgr, frame_cnt = zero in first frame.

---
22 5F 4B canB_iso_rcv_mgr, pci_cnt = 0 in first frame.

---
22 5F 4C canB_iso_rcv_mgr, pci_cnt > 8192 in first frame.

---
22 5F 4D canB_iso_rcv_mgr, no buffer available.

---
22 5F 4E canB_iso_rcv_mgr, flow control frame too short.

---
22 5F 4F canB_iso_rcv_mgr, buffer not expecting flow control frame.

---
22 5F 50 canB_iso_rcv_mgr, invalid separation time in flow control frame, $80 to $F0.

---
22 5F 51 canB_iso_rcv_mgr, invalid separation time in flow control frame, > $FA.
Set to 1 msec.

---
22 5F 52 canB_iso_rcv_mgr,invalid flow status in flow control frame.

---
22 5F 53 canB_iso_rcv_mgr, unknown frame type.

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

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

---
22 5F 56 canB_iso_rcv_buff_mgr, buffer time-out, first frame received,
flow control transmit pending.

---
22 5F 57 canB_iso_rcv_buff_mgr, buffer time-out while receiving data.

---
22 5F 58 canB_iso_rcv_buff_mgr, time-out waiting to send buffer to client.

---

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, 11-bit ID, no ae, transmit command too short.

---

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

---

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

---

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

---

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

---

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

---

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

---

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

---

22 5F 63 canB_iso_xmt_proc, data count > 4095 and EDL is false.

---

22 5F 64 canB_iso_xmt_proc, invalid transmit case.

---

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.

---

22 5F 68 canB_iso_xmt_buff_mgr, buffer state = 12, watchdog expired.

---

22 5F 69 canB_iso_xmt_buff_mgr, buffer state = 13, watchdog expired.

---

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

---

22 5F 6B canB_iso_xmt_buff_mgr, buffer state = 13 buffer time-out.

---

22 5F 6C canB_iso_xmt_buff_mgr, buffer state = 13, buffer time-out.

---

22 5F 6D canB_iso_xmt_buff_mgr, buffer state = 14, buffer time-out.

---

22 5F 6E canB_iso_buff_mgr, invalid channel number.

---

22 5F 6F canB_iso_buff_mgr, invalid buffer number.

---

22 5F 70 canB_iso_buff_mgr, invalid buffer state.

---

22 5F 71 canB_iso_buff_mgr, invalid buffer state.

---

22 5F 72 canB_iso_buff_mgr, invalid object number.

---

22 5F 73 canB_iso_buff_mgr, invalid mate entry.

---

22 5F 74 canB_iso_xmt_proc_FD, error in ‘nearest dlc’, single frame, type1, no ae, padding disabled.

---

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

---

22 5F 76

---

22 5F 77

---

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.

---

22 5F 81 canB_iso_xmt_buff_mgr, bad dlc decode, buffer state = 13, no ae.

---

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

---

22 5F 83 canB_iso_buff_mgr: invalid obj_buff entry.

---

22 5F 84

---

22 5F 85

---

22 5F 86

---
22 5F 87

---
22 5F 88

---
22 7F 00

---
22 7F 01

---
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 '0x' transmit command, 11-bit, command too short.

---
22 7F 07 '0x' transmit command, 11-bit, command too long.

---
22 7F 08 '0x' 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 Transmit command, data length too short for edl = 9 (12 data bytes)
0x transmit command, 11-bit ID.
CAN2 or CAN3 only.

---

22 7F 11 Transmit command, data length too short for edl = 9 (12 data bytes)
0x transmit command, 29-bit ID.
CAN2 or CAN3 only.

---

22 7F 12 Improper 0x transmit command.
Can't have (EDL or BRS) true and RTR true.

---

22 7F 13 Improper 0x transmit command.
DLC > 8 and EDL flag is false.

---

22 7F 14 Improper 12 transmit command.
Can't have (EDL or BRS) true and RTR true.

---

22 7F 15 Improper 12 transmit command.
Can't have BRS true with EDL false.

---

22 7F 16 Improper 12 transmit command.

DLC > 8 and EDL flag is false.

---

22 7F 17 Improper 0x transmit command.
Can't have BRS true with EDL false.

---

22 7F 20 Object is receive busy in ’74 07’ command.

---

22 7F 25 canA_rcv_mgr, invalid channel number.

---

22 7F 26 canA_rcv_mgr, dlc > 8.

---

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.

---

22 7F 2B canB_xmt_ack_mgr, invalid channel number.

---

22 7F 2C canB_error_mgr, invalid channel number.

---

22 7F 2D canB_rcv_mgr, a 'return(3)' from canB_iso_rcv_mgr.

---

22 7F 2E

---

22 7F 2F

---

21 84 Command buffer mode fault.

---

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.

---

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: edl is clear, dlc > 8.

---

22 92 0z

CANz bus off warning.
z: CAN channel 0, 1, 2, 3.

---

24 93 0z vv ww

CANz lost frame counter.

z: CAN channel 0, 1, 2, 3.

vv ww: lost frame counter.

---

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.

---

24 97 0z rr tt

Classical CAN error counters.

z: CAN channel 2, 3.

rr: receive error counter.

tt: transmit error counter.

---

26 98 0y rr ss tt vv

CANy-FD error counter ii

This error response is triggered if any counter overflows

$7F to $80 is overflow.

y: CAN channel 2 or 3

rr: transmit errors during data phase (fast baud rate)

ss: transmit errors during arb phase (slow baud rate)

tt: receive errors during data phase (fast baud rate)

vv: receive errors during arb phase (slow baud rate)

---

22 E5 01 LIN0 transmit command too short.

---

22 E6 01 KWP transmit command too short.

---

22 E7 01 LIN1 transmit command too short.

---

22 E9 01 Flexray transmit command too short.

3: Command error.

---

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.

4:

5:

6: Configuration reports.

LIN0, LIN1

63 01 0r 0z Send received checksum to client.
r: channel: 7, 5.
z: 0: disabled.
1: enabled.

LIN0, LIN1

63 02 0r zz Receiver buffer timeout.
r: channel: 7, 5.
zz: time in milliseconds.

CAN0, CAN1, CAN2, CAN3, LIN0, LIN1, KWP

63 08 0r 0y: Time stamp status.
- r: channel: 0, 1, 2, 3, 7, 5, 6.
- y: 0 disabled.
  1 enabled.

LIN0, LIN1

64 1D 0r yy zz: Synch break time.
- r: channel: 7, 5.
- yy zz: increment count. (increment = 1.024 usec).

CAN0, CAN1, CAN2, CAN3, LIN0, LIN1, KWP

63 40 0r 0y: Send transmit acknowledgements to client.
- r: channel: 0, 1, 2, 3, 7, 5, 6.
- y: 0 disabled.
  1 enabled.

LIN0, LIN1

63 50 0r 0z: Baud rate.
- r: channel: 7, 5.
- z: 1: 2400 baud.
  2: 9600 baud.
  3: 19200 baud.
  4: 10400 baud.

64 50 0r yy zz: Baud rate.
- r: channel: 7, 5.
- yy zz: divisor load.
  125,000,000 / 32 / yy zz
  (where 'yy zz' is converted to decimal)

LIN0, LIN1

63 52 0r zz: Maximum frame time.
- r: channel: 7, 5.
- zz: time in milliseconds.

LIN0, LIN1

63 5A 0r 0z: Checksum type.
- r: channel: 7, 5.
- z: 0: classic (LIN 1.3).

1: enhanced (LIN 2.0).
2: none.

LIN0, LIN1

63 66 0r 0z "ID byte only" error message to client.
r: channel: 7, 5.
z: 0: disabled.
1: enabled.

LIN0, LIN1

63 69 0r 0z Secondary operations.
r: channel: 7, 5.
z: 0: disabled.
1: enabled.

---

63 6A yy zz: CPU heart beat LED blink rate.
yy zz: LED half period time, msec.

7: Initialization attempt response.

8: CAN configuration reports.

CAN0, CAN1, CAN2, CAN3

83 04 0r 0y Configuration of CAN object.
r channel: 0, 1, 2, 3.
y: 0: object disabled.
1: object enabled for receive.
2: object enabled for transmit. (CAN0/1 only.)

CAN0, CAN1

85 05 0r yz ss tt: Configuration of 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 05 0r yz ss tt vv ww: Configuration of 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 (Receive object only.)

85 05 0r yz ss tt: Object 11-bit ID.
- r: channel: 2, 3.
- y: b7: 0.
  - b6: 0.
  - b5: EDL.
    - 0: Classical CAN frame.
    - 1: CAN-FD frame.
  - b4: 0.
- z: object number: $0 to $F.
- ss tt: 11-bit ID, right justified.

87 05 0r yz ss tt vv ww: Object 29-bit ID.
- r: channel: 2, 3.
- y: b7: 0.
  - b6: 0.
  - b5: EDL.
    - 0: Classical CAN frame.
    - 1: CAN-FD frame.
  - b4: 0.
- z: object number: $0 to $F.
- ss tt vv ww: 29-bit ID, right justified.

CAN0, CAN1

8x 06 0r 0z rr ss tt …: Object data.
- r: channel: 0, 1, 2, 3.
- z: object number: $0 to $F.
- rr ss tt …: data

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.

CAN0, CAN1, CAN2, CAN3

83 08 0r 0w: Transit acknowledgement status.
- r: channel: 0, 1, 2, 3.
- w: 0: transmit acks disabled.
  1: transmit acks enabled.

CAN0, CAN1

83 0A 0r yy: Baud rate.
- r channel: 0, 1.
- yy: 00: User 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.
- yy: 00: User 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.
- zz: 00: User 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 CAN0 and CAN1 bit definitions.

Contact the factory for a complete definition of the bit timing registers for CAN2 and CAN3.

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.

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: 0 = is not an FD frame (EDL bit = 0)
  1 = is an FD frame (EDL bit = 1)
  b4: 0 = data at regular speed (BRS bit = 0)
  1 = data at high speed (BRS bit = 1)
- 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: 0 = is not an FD frame (EDL bit = 0)
1 = is an FD frame (EDL bit = 1)
b4: 0 = data at regular speed (BRS bit = 0)
1 = data at high speed (BRS bit = 1)
z: object number: $0 to $F.
tt vv mm nn: 29-bit ID, right justified.

NOTE: The periodic message set-up command (7x 18) format is different from the AVT-853 command set.

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 non-FD frame).
0: normal frame.
1: RTR true, remote transmit request.
b5: EDL.
0: EDL bit: Classical CAN frame.
1: EDL bit: CAN-FD frame.
b4: BRS.
0: BRS bit: data field at normal speed.
1: BRS bit: data field 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.

LIN0, LIN1

8x 18 0r pp 0m rr ss tt … Periodic message set-up.
- r: channel: 7, 5.
- pp: message number, $00 to $0F.
- m: 0: slave.
  1: master.
- rr ss tt ...: data field. [optional]

KWP

8x 18 06 pp rr ss tt … Periodic message set-up.
- channel: 6.
- pp: message number, $00 to $0F.
- rr ss tt ...: 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 zz 0v: Periodic message disable/enable status.
- r: channel: 0, 1, 2, 3.
- zz: message number, $00 to $2F.
- v: 0 disabled.
  1 enabled.

LIN0, LIN1, KWP

84 1A 0r zz 0v: Periodic message disable/enable.
- r: channel: 7, 5, 6.
- zz: message number, $00 to $0F.
- v: 0 disabled.
  1 enabled.

CAN0, CAN1, CAN2, CAN3

85 1B 0r zz vv ww: Periodic message interval count.
- r: channel: 0, 1, 2, 3.
- zz: message number, $00 to $2F.
- ww vv: interval count.

LIN0, LIN1, KWP

85 1B 0r zz vv ww: Periodic message interval count.
- r: channel: 7, 5, 6.

zz: message number, $00 to $0F.
ww vv: time in milliseconds.

CAN0, CAN1, CAN2, CAN3, LIN0, LIN1, KWP

62 1C 0r All periodic message for channel 'r' disabled.
r: channel: 0, 1, 2, 3, 7, 5, 6.
r: 'F' – all messages, all channels disabled.

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.

CAN0, CAN1

85 2C 0r 0z ss tt: 11-bit mask.
r: channel: 0, 1.
z: mask number, $0 to $F.
ss tt: mask value, 11-bit.

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

87 2C 0r 0z ss tt vv ww: 29-bit mask.
r: channel: 0, 1.
z: mask number, $0 to $F.
ss tt vv ww: mask value, 29-bit.

CAN2, CAN3

85 2C 0r yz ss tt: 11-bit mask.
r: channel: 2, 3.
y: b7: IDE bit.
b6: 0.
b5: EDL bit.
b4: 0
z: mask number, $0 to $F.
ss tt: mask value, 11-bit.
1: bit must match.
0: bit is don't care.

87 2C 0r yz ss tt vv ww: 29-bit mask.
- r: channel: 2, 3.
- y: b7: IDE bit.
  - b6: 0.
  - b5: EDL bit.
  - b4: 0
- 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 0s: Disable / Enable ISO 15765 'AE' operation.
- r: CAN channel 0, 1, 2, 3.
- z: Object number 0 to F.
- s: 0 = disabled, 1 = enabled.

85 30 0r 0z 0s ww: Disable / Enable ISO 15765 'AE' operation; specify 'AE' byte.
- r: CAN channel 0, 1, 2, 3.
- z: Object number 0 to F.
- s: 0 = disabled, 1 = enabled.
- ww: 'ae' byte.

CAN1

82 45 01: Set CAN1 transceiver to single wire CAN (SWC).

82 45 02: Set 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.

CAN2, CAN3

84 63 0r ss tt: Transmit attempt limit.
r: channel: 2, 3.
ss tt: number of attempts before message is discarded.
a value of 00 00 means indefinite (continuous, no limit).

9: Board status information.

92 01 10: CAN0 reset complete.

92 01 11: CAN1 reset complete.

92 01 12: CAN2 reset complete.

92 01 13: CAN3 reset complete.

92 01 20: LIN0 reset complete.

92 01 21: LIN1 reset complete.

92 01 30: FLEXRAY hardware reset complete.

93 04 xx yy: Firmware version report. Version is xx yy.

93 28 0x yz: Model number report. xyz is the model number.

91 3A: AVT-423 Ethernet connect and operating report.

94 3B xx yy zz: FPGA version number report.

A:

B:

C:

D:

E:

F:

16. Appendix A

xxx

17. Appendix B

Xxx

Change / Version Notes

0003: Initial release.

0004: Changed LIN0 to channel 7 and LIN1 to channel 5. This puts LIN0, LIN1, and KWP into the same associations as the AVT-853. Other minor corrections in the command and responses sections.

0005: LIN0 and LIN1 are now operational. Corrected and updated various sections. Reviewed, updated, and corrected some commands and responses.

0006: Many changes to channel byte of CAN transmit and periodic message configuration commands and receive response format.

Changed how RTR, EDL, and BRS bits are handled as well as how the CAN frames are configured.

Increased the number of CAN periodic messages to 30 (decimal). Index numbers $00 to $2F.

Added text for LIN1. LIN0 references LIN1, as they are nearly identical.

Lots of other corrections and (hopefully) better operational explanations.

5 July 2017: Corrected the status responses on page 68.

0009: This version of firmware was a major upgrade. Lots of corrections to commands and responses, major change in how data moving to the client is handled, at least one error response was corrected, one error response was added, one error response was removed.

Manual changes were pretty much limited to corrections.

3 August 2017: Correction to the ’26 98 …’ error response.

29 August 2017: Correction to the ’75 1B’ command and ’85 1B’ response.
(For LIN and KWP it was incorrectly listed as ’74 1B’ and ’84 1B’.

0012: A major upgrade. Many routines were cleaned up and changed to improve code efficiency, maintainability, and speed. All CAN0 and CAN1 variables were changed to ‘structures’. Message handling using ISO 15765 was added for channels CAN0 and CAN1. Method used to write to Fifo2 (communications to the client) was fundamentally changed to improve speed.

The following commands (that existed in previous versions) were significantly changed:

7x 2A.
7x 2C.
0x transmit.
11 xx transmit.
12 xx yy transmit.

The following commands were added to support ISO 15765 functionality.

7x 05.
7x 06.
7x 07.

7x 08.
7x 0E.
7x 27.
7x 28.

0013 Found and corrected errors in the CAN set Acceptance ID commands: ('7x 05' and '7x 2A').

Found and corrected errors in the CAN set Acceptance ID Mask command: ('7x 2C').

Updated the manual with a note that the user should set the CAN Acceptance ID 'before' setting the CAN Acceptance ID Mask. The order does not affect operations, but will affect the 'look' of the mask response.

Updated the manual with a note that the CAN Acceptance ID commands '7x 05' and '7x 2A' are identical and redundant. The user can use either one, as they please.

Corrected typo on page 59; "or" should be "0r".

0019 Removed all references to BroadR-Reach – that project has been terminated.

Added the '24 97 0y rr tt' error response.

0019 (B) Corrected the '7x 0A' baud rate command. Added option "E" for 5 Mbaud.

Added note to '7x 0A' baud rate command that channels 2 and 3 will always report both low and high speed baud rates for either form of baud rate command.
In other words: '73 0A' and '74 0A' commands both result in '84 0A' response.

0021 (A) Added all the commands and notes for CAN2 and CAN3 ISO15765 support.

0021 (B) A bunch of "canB_" error messages were incorrectly identified as "22 7F yy". They were corrected to "22 5F yy" messages.

0023 (A) Added a note about trying to use 'blue' text to indicate changes from previous manual version.
Added note about transmit attempt limit for channels CAN2 and CAN3.
Added '7x 63' command and '8x 63' response.

0025 (A) Added the '7x 29' command.
Added notes about setting "max_dlc" value when using ISO15765 with CAN-FD channels.

18. Questions ??

Contact the factory by e-mail, phone, or fax.
Contact information is provided here and on the bottom of page 1.

Post: 1509 Manor View Road
Davidsonville, MD 21035 USA

Phone: +1-410-798-4038
E-mail: Support@AVT-HQ.com
Web site: www.AVT-HQ.com

AVT-423 specific web pages:

Set IP address
Update Operation Software
Operation Firmware file
http://www.AVT-HQ.com/dowload.htm#AVT-423

Operation Firmware Version Descriptions
http://www.AVT-HQ.com/423_sw.htm