



**ADVANCED VEHICLE TECHNOLOGIES, Inc.**

---

## AVT - 423

### Multiple Interface

CAN0: 2-wire (high speed) (Classical only; non-FD)

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

CAN2: 2-wire (high speed) (FD and Classical)

CAN3: 2-wire (high speed) (FD and Classical)

LIN0: LIN communications

KWP0: Keyword Protocol K-Line communications

LIN1: LIN communications

KWP1: Keyword Protocol K-Line communications

AVT-424 LIN Expansion board (optional)

6 additional channels of LIN

~~Flexray: Dual or Single channel communications~~

~~(not installed, not functional)~~

Firmware Version 00 68  
19 November 2022

# Table of Contents

<b>1. INTRODUCTION .....</b>	<b>6</b>
1.1 BLUE NOTES .....	6
1.2 NEW STUFF .....	6
1.3 HARDWARE.....	7
1.3.1 <i>Summary of Hardware Updates</i> .....	7
1.4 FIRMWARE CAPABILITIES .....	7
1.4.1 CAN0.....	7
1.4.2 CAN1.....	7
1.4.3 CAN2.....	7
1.4.4 CAN3.....	7
1.4.5 LIN0 or KWPO .....	7
1.4.6 LIN1 or KWPI .....	8
1.4.7 LIN2 thru LIN7 .....	8
1.5 FIRMWARE NOTE #1.....	8
1.6 FIRMWARE NOTE #2.....	8
1.7 FIRMWARE NOTE #3.....	8
1.8 FIRMWARE NOTE #4.....	8
1.9 FIRMWARE NOTE #5.....	9
1.10 FIRMWARE NOTE #6.....	9
1.11 FIRMWARE NOTE #7.....	9
1.12 FIRMWARE NOTE #8.....	9
1.13 FIRMWARE NOTE #9.....	9
1.14 FIRMWARE NOTE #10.....	9
1.15 FIRMWARE PLAN.....	9
1.16 FIRMWARE .....	10
1.16.1 <i>Determining Firmware Version</i> .....	10
1.16.2 <i>Determining Model Number</i> .....	10
1.16.3 <i>Determining Board Revision Level</i> .....	10
COMMANDS AND RESPONSES .....	10
<b>2. GLOSSARY .....</b>	<b>10</b>
<b>3. AVT-423 OPERATION .....</b>	<b>12</b>
<b>4. AVT-423 CPU .....</b>	<b>12</b>
4.1 SUPPORT SOFTWARE .....	12
<b>5. CLIENT COMPUTER CONNECTION .....</b>	<b>13</b>
5.1 AVT-423 CONNECTION TO CLIENT COMPUTER .....	13
5.1.2 <i>Ethernet IP Addressing Modes</i> .....	14
5.1.3 <i>Changing the IP Address</i> .....	14
5.2 MULTIPLE CLIENT CONNECTIONS.....	14
5.2.1 <i>Multi-Client Operation Description</i> .....	15
5.3 PACKET COMMUNICATIONS BETWEEN THE AVT-423 AND THE CLIENT COMPUTER .....	15
<b>6. POWER AND NETWORK CONNECTION .....</b>	<b>16</b>
6.1 AVT-423 CONNECTIONS - BOARD REVISIONS “B”, “C”, AND “D” .....	17
6.2 AVT-423 CONNECTIONS - BOARD REVISION “F”.....	18
6.3 POWER REQUIREMENTS .....	19
6.3.1 <i>Ground</i> .....	19
6.3.2 <i>Input Voltage</i> .....	19
6.3.3 <i>Power Dissipation</i> .....	19
6.3.4 <i>Fuse</i> .....	20

<b>7. ADC CONNECTION .....</b>	<b>20</b>
<b>8. DAC CONNECTION .....</b>	<b>20</b>
<b>9. DIGITAL OUTPUT .....</b>	<b>20</b>
9.1 DIGITAL OUTPUT CONNECTION .....	20
9.2 DIGITAL OUTPUT OPERATION.....	20
<b>10. OPERATION MODES .....</b>	<b>21</b>
10.1 SIMULTANEOUS NETWORK OPERATIONS.....	21
10.2 START-UP PARAMETERS STORED IN NON-VOLATILE MEMORY .....	21
10.3 THE DIFFERENT ‘AUTOMATIC’ RESPONSES; AVT-423 TO CLIENT .....	22
10.4 TYPES OF RESETS.....	22
10.4.1 The ‘F1 A5’ Reset.....	22
10.4.2 The ‘F1 C3’ Reset.....	22
<b>11. NETWORK HARDWARE DESCRIPTIONS.....</b>	<b>23</b>
11.1 CAN0 - 2-WIRE CAN .....	23
11.1.1 CAN0 Channel Number .....	23
11.2 CAN1 - 2-WIRE CAN OR SINGLE WIRE CAN.....	23
11.2.1 CAN1 Channel Number .....	24
11.3 CAN2 - 2-WIRE CAN .....	24
11.3.1 CAN2 Channel Number .....	24
11.4 CAN3 - 2-WIRE CAN .....	24
11.4.1 CAN3 Channel Number .....	25
11.5 LIN0 .....	25
11.5.1 LIN0 Channel Number .....	25
11.6 LIN1 .....	25
11.6.1 LIN1 Channel Number .....	25
11.7 KWP0.....	25
11.7.1 KWP0 Channel Number .....	25
11.8 KWP1.....	25
11.8.1 KWP1 Channel Number .....	26
11.9 LIN2 THRU LIN7.....	26
11.10 LIN2 THRU LIN7 .....	26
11.10.1 LIN2 thru LIN7 Channel Numbers .....	26
11.11 FLEXRAY .....	26
11.11.1 Flexray Channel Number .....	26
<b>12. CAN CHANNEL OPERATIONS.....</b>	<b>26</b>
12.1 CAN AND CAN-FD NOTES .....	26
12.2 CAN-FD IP CORE NOTES .....	27
12.2.1 Version 0004.....	27
12.2.2 Version 0010.....	27
12.3 CAN CHANNEL OPERATIONAL MODES .....	27
12.3.1 Disabled.....	28
12.3.2 Normal.....	28
12.3.3 Listen Only .....	28
12.3.4 Transmit Command .....	28
12.3.5 Receive Response.....	29
12.3.6 Time Stamps .....	30
12.4 OBJECT ID AND MASK .....	31
12.4.1 Configuration – CAN0 and CAN1 .....	31
12.4.2 Configuration – CAN2 and CAN3 .....	31
12.4.3 Object ID and Mask Operation .....	32
12.4.4 Acceptance ID and Mask Notes.....	32

12.4.5	CAN0 and CAN1 specifics.....	32
12.4.6	CAN2 and CAN3 specifics.....	33
12.5	SETTING UP CAN0 OR CAN1 FOR OPERATION .....	33
12.5.1	Communications Example.....	34
12.6	SETTING UP CAN2 OR CAN3 FOR OPERATION .....	34
12.6.1	Classical CAN Communications Example.....	34
12.6.2	CAN-FD Communications Example.....	35
12.6.3	Transmit Attempt Limits .....	35
12.7	PERIODIC MESSAGE SUPPORT .....	36
12.7.1	Periodic Message Number and Numbering.....	36
12.7.2	Periodic Message Data Field Length.....	36
12.7.3	Long Periodic Message Definition and Response.....	36
12.7.4	Type1 Periodic Messages.....	37
12.7.5	Type2 Periodic Messages.....	38
12.7.6	Periodic Message Commands .....	38
12.8	PERIODIC MESSAGE SPECIAL FUNCTIONS.....	38
12.8.1	CAN Frame Data Definition.....	38
12.8.2	Special Function: RC6.....	39
12.8.3	Special Function xxx.....	39
12.8.4	Special Function xxx.....	39
12.9	PERIODIC PAUSE FUNCTION .....	39
12.9.1	Periodic Pause Function Command.....	39
12.10	FRAME BUFFERS.....	39
12.10.1	Frame Buffer Description.....	39
12.10.2	Frame Buffer FIFO Mode .....	40
12.10.3	Frame Buffer FIFO Mode Notes .....	41
12.10.4	Frame Buffer Rolling Mode.....	41
12.10.5	Frame Buffer Rolling Mode Notes.....	41
12.10.6	Frame Buffer General Notes .....	42
12.10.7	Frame Buffer Command Summary.....	42
12.11	ISO 15765 SUPPORT FOR CAN0 AND CAN1 .....	43
12.11.1	Basic Set-up and Operational Discussion.....	43
12.11.2	Differences with ISO 15765 on CAN0/1 (as compared to CAN2/3).....	44
12.11.3	ISO 15765 Initialization and Operation Discussion .....	44
12.12	ISO 15765 SUPPORT FOR CAN2 AND CAN3 .....	46
12.12.1	Basic Set-up and Operational Discussion.....	46
12.12.2	Differences with ISO 15765 on CAN2/3 (as compared to CAN0/1).....	48
12.12.3	ISO 15765 Initialization and Operation Discussion – Classical CAN.....	48
12.12.4	ISO 15765 Initialization and Operation Discussion –CAN-FD .....	50
12.13	CAN ‘MIRROR’ FUNCTION .....	52
12.13.1	Operational Description.....	52
12.13.2	Commands .....	52
<b>13.</b>	<b>LIN1 OPERATIONS .....</b>	<b>53</b>
13.1	COMMUNICATIONS.....	53
13.2	LIN1 BUS SUPPLY VOLTAGE .....	53
13.3	LIN1 BUS SUPPLY – SELECTING A SEPARATE EXTERNAL SUPPLY .....	53
13.3.1	Message Length.....	54
13.3.2	Checksum.....	54
13.3.3	ID Byte Only Message .....	54
13.3.4	Communications Example .....	55
13.3.5	Time Stamp.....	56
13.4	SPECIAL FUNCTIONS.....	56
13.4.1	LIN Frame Data Definition .....	57
13.4.2	LIN Mirror Function .....	57
13.4.3	Special Function 1.....	57

---

13.5	PERIODIC MESSAGE SUPPORT .....	58
13.5.1	<i>LIN Frame Data Definition</i> .....	58
13.5.2	<i>Modes of Operation</i> .....	58
13.5.3	<i>Organization of Periodic Messages</i> .....	58
13.5.4	<i>Type1 Periodic Message</i> .....	58
13.5.5	<i>Slave Periodic Message</i> .....	59
13.5.6	<i>Periodic Message Commands</i> .....	59
13.6	PERIODIC MESSAGE SPECIAL FUNCTIONS.....	59
13.6.1	<i>LIN Frame Data Definition</i> .....	59
13.6.2	<i>Periodic Message Special Function 0</i> .....	60
13.7	ABIC SUPPORT .....	60
13.8	COMMANDS AND RESPONSES .....	60
<b>14.</b>	<b>LIN0 OPERATIONS .....</b>	<b>60</b>
14.1	LIN0 OPERATION NOTES .....	60
14.2	LIN0 BUS SUPPLY VOLTAGE .....	60
14.3	LIN0 BUS SUPPLY – SELECTING A SEPARATE EXTERNAL SUPPLY .....	61
14.4	COMMANDS AND RESPONSES .....	61
<b>15.</b>	<b>KWP1 OPERATIONS.....</b>	<b>61</b>
15.1	KWP1 OPERATION LIMITATIONS .....	61
15.2	KWP1 BUS SUPPLY VOLTAGE .....	61
15.3	KWP1 K-LINE SUPPLY – SELECTING A SEPARATE EXTERNAL SUPPLY.....	62
15.4	COMMANDS AND RESPONSES .....	62
15.5	KWP1 OPERATION – QUICK INTRO.....	62
15.6	KWP1 OPERATION – FAST INIT.....	63
<b>16.</b>	<b>KWP0 OPERATIONS.....</b>	<b>64</b>
16.1	KWP0 OPERATION LIMITATIONS .....	64
16.2	KWP0 BUS SUPPLY VOLTAGE .....	64
16.3	KWP0 K-LINE SUPPLY – SELECTING A SEPARATE EXTERNAL SUPPLY.....	64
16.4	COMMANDS AND RESPONSES .....	65
16.5	KWP0 OPERATIONS.....	65
<b>17.</b>	<b>COMMANDS .....</b>	<b>66</b>
17.1	RESPONSES .....	100
<b>18.</b>	<b>APPENDIX A - AVT-424 LIN EXPANSION BOARD.....</b>	<b>156</b>
<b>19.</b>	<b>APPENDIX B .....</b>	<b>159</b>
<b>20.</b>	<b>QUESTIONS ??.....</b>	<b>164</b>

## 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 only.  
channel 5.
- KWP0: Keyword Protocol (ISO 14230) only.  
channel 8.
- KWP1: Keyword Protocol (ISO 14230) only.  
channel 6.
- LIN2 thru 7: LIN only.  
channels \$A thru \$F  
Only present when the AVT-424 LIN Expansion board is installed.
- Flexray: Dual or Single channel.  
Not installed. Not functional.  
No intention of ever implementing Flexray. It was designated channel 9.

All operations are simultaneous with the following exceptions.

LIN0 and KWP0 are mutually exclusive – they share one hardware channel.

LIN1 and KWP1 are mutually exclusive – they share one hardware channel.

### 1.1 Blue Notes

I try to put all new and updated text in 'blue'.

'Blue' text will indicate changes from the previous (major) version of the manual.

### 1.2 New Stuff

The AVT-424 LIN Expansion board is now available.

It is a (relatively) small board that is installed on top of an AVT-423 board. It provides an additional 6 channels of LIN. Firmware version '0048' (and later) fully supports the AVT-424.

Refer to Appendix “A” for detailed information about the AVT-424.

### **1.3 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](http://www.AVT-HQ.com/423_hw.htm)

#### **1.3.1 Summary of Hardware Updates**

The following changes were implemented in hardware revision ‘F’.

JP1 was added to allow the user to connect external and separate power supplies to the transceiver circuitry for LIN0/KWP0.

JP2 was added to allow the user to connect external and separate power supplies to the transceiver circuitry for LIN1/KWP1.

Circuitry and board-to-board connectors were added to install the AVT-424 LIN Expansion board on top of the AVT-423 board.

### **1.4 Firmware Capabilities**

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

#### **1.4.1 CAN0**

CAN (so-called Classical or non-FD). Transmit and receive. Sixteen objects (either receive or transmit). [Thirty two \(32 decimal\) periodic messages](#). Periodic messages are Type1 only.

#### **1.4.2 CAN1**

CAN (so-called Classical or non-FD). Transmit and receive. Sixteen objects (either receive or transmit). [Thirty two \(32 decimal\) periodic messages](#). Periodic messages are Type1 only.

#### **1.4.3 CAN2**

Classical CAN and CAN-FD capable. Transmit and receive. Sixty-Four receive objects. Sixty-Four transmit objects. [Thirty two \(32 decimal\) periodic messages](#). Periodic messages are Type1 only. ISO CAN frame CRC (non-ISO possibly available). Maximum data payload of 64 bytes supported. Maximum baud rate of 8 Mbaud supported.

#### **1.4.4 CAN3**

Classical CAN and CAN-FD capable. Transmit and receive. Sixty-Four receive objects. Sixty-Four transmit objects. [Thirty two \(32 decimal\) 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.

#### **1.4.5 LIN0 or KWP0**

LIN or KWP operations including receive, transmit, and periodic messages.

#### **1.4.6 LIN1 or KWP1**

LIN or KWP operations including receive, transmit, and periodic messages.

#### **1.4.7 LIN2 thru LIN7**

Six more channels of LIN operations when the AVT-424 LIN Expansion board is installed.

#### **1.5 Firmware Note #1**

[This paragraph was removed with manual version '0058'.](#)

#### **1.6 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 necessary. This was all done in an effort to make the AVT-423 command set 'friendly' to users familiar with the AVT-718.

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

#### **1.7 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 12.12 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.

#### **1.8 Firmware Note #4**

As of version '0043' the '7x 17' command has been updated to support the 64 transmit and 64 receive objects of CAN2 and CAN3. There are two new forms of the command that must be used to access all 64 transmit and receive objects.

As of version '0043' the '7x 2A' command has been updated to support the 64 transmit and 64 receive objects of CAN2 and CAN3. There are two new forms of the command that must be used to access all 64 transmit and receive objects.

As of version '0043' the '7x 2C' command has been updated to support the 64 transmit and 64 receive objects of CAN2 and CAN3. There are two new forms of the command that must be used to access all 64 transmit and receive objects.

The following commands have been updated to support all the CAN2 and CAN3 objects.

7x 04

7x 27

7x 28

7x 29

As of version '0040' the '7x 05' command is now deprecated. It will no longer be updated and will likely be removed at some point in the future.

### **1.9 Firmware Note #5**

As of version '0043' limited Keyword Protocol (KWP), ISO 14230, functions have been implemented. Both hardware K-line channels can be operated to support either LIN or KWP communications.

### **1.10 Firmware Note #6**

As of version '0048'.

The AVT-424 LIN Expansion board now fully supported.

Refer to Appendix "A" for information about the AVT-424 LIN Expansion board.

### **1.11 Firmware Note #7**

As of version '0049'.

The following additional Keyword Protocol (KWP, ISO 14230) functions have been implemented.

Periodic message support for both KWP channels.

Fast Init support for both KWP channels.

### **1.12 Firmware Note #8**

As of version '0054'.

Support was added for 'long' periodic messages for channels CAN2 and CAN3. This means periodic messages with data field lengths up to and including 64 bytes can be defined.

### **1.13 Firmware Note #9**

As of version '0056'.

The maximum number of CAN periodic messages was reduced to 32 (decimal) for CAN channels 0, 1, 2, and 3. (Previously the maximum was 48, decimal.)

All CAN periodic messages are numbered: 0x00 to 0x1F.

### **1.14 Firmware Note #10**

As of version '0068'.

The following commands have been removed:

### **1.15 Firmware Plan**

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

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

## 1.16 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](http://www.AVT-HQ.com/432_sw.htm)

### 1.16.1 Determining Firmware Version

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

- Power-on the AVT-423 interface unit.
- Connect to a Client computer running the Hex Terminal or equivalent.
- The connect 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.

### 1.16.2 Determining Model Number

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

- Power on the AVT-423 interface unit.
- Connect to a Client computer running the Hex Terminal or equivalent.
- The connect notification is:  
    \$91 \$3A        indicates AVT-423 operations.  
    \$93 \$04 \$xx \$yy where 'xx yy' is the firmware version.
- Send the \$B1 03 command.
- The response will be: \$93 \$28 \$xx \$yy where xxyy forms the model number.  
(eg. 04 23)

### 1.16.3 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 11898-2	ISO specification for high speed 2-wire CAN physical layer.
ISO 15765	An ISO specification dealing with the formatting of data in the CAN frame data field. Also used in sending blocks of data using CAN. Also known as Multi-Frame Messaging (MFM) or Segmented Messages. This specification also deals with other CAN network issues.
J2411	An SAE specification for Single Wire CAN (SWC).
K-line	Single wire communications protocol. Refer to ISO 9141, ISO 9141-2, and ISO 14230 for more information.
KWP	Key Word Protocol. Several versions exist, the most common being Key Word Protocol 2000, which is ISO 14230. This is a ‘superset’ of ISO 9141 and ISO 9141-2.
KWP0	KWP, channel 8. Uses pin # 23.
KWP1	KWP, channel 6. Uses pin # 11.
LIN	Local Interconnect Network.
LIN0	LIN, channel 7. Uses pin # 23.
LIN1	LIN, channel 5. Uses pin # 11.
LIN2 thru LIN7:	LIN, channels \$A thru \$F. Refer to Appendix “A”.

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.

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 3 seconds.  
(Mike - xxx – this needs to be verified.)

### 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 Firmware 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.

[If you want an old version of firmware – send me an email. I archive all previous versions.](#)

## **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 send the ‘B1 04’ command to request the MAC address.

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.

### **5.1.1.3 TCP/IP Port**

Communications with the AVT-423 vehicle network interface is via port # 10001, 10002, or 10003. Refer to Section 5.2 for more information about multiple Client connections.

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 Multiple Client Connections**

For firmware versions 0028 and prior: Only one Client connection was supported. The only connection allowed was to TCP port # 10001.

For firmware version 0030 and later – three simultaneous Client connects are supported.

[For firmware version 0059 and later – four simultaneous Client connects are supported.](#)

All four connections are to the same IP address. But they use different port numbers.

The three available Client connection ports are:

TCP port # 10001.

TCP port # 10002.

TCP port # 10003.

TCP port # 10004.

All three connections operate identically, as described in the next section

### 5.2.1 Multi-Client Operation Description

A Client can connect to any available port.

Only one Client connection per port.

When more than one Client is connected to the AVT-423 and when processing commands from a Client - port # 10001 has higher priority than port # 10002. Similarly port # 10002 as higher priority than port # 10003. Specifically this means that if two or more commands are received at exactly the same time, the command from the lowest numbered port will be processed first. That is all 'higher priority' means; in this context. Otherwise, all Clients (ports) are treated equally.

**Important Note:** All responses are sent out all active ports. This is an important point. For example, if the Client on port # 10002 sends the 'B1 01' firmware version request, the AVT-423 will send the '93 04 xx yy' response out all active ports. That means a Client that did NOT send that command will still receive the response.

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

### 5.3 Packet Communications Between the AVT-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 yyThese 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.

---

## AVT-423 Multiple Interface

---

\$xx indicates the count of bytes to follow (not including the xx byte).  
\$rr ss tt ... the packet data, including the message to/from the network.

- If the byte count is more than \$FF but less than or equal to \$FF FF the packet will be of the form: \$12 xx yy rr ss tt  
\$12 indicates second alternate header format.  
\$xx yy indicates the count of bytes to follow (not including the xx yy bytes).  
\$rr ss tt ... the packet data, including the message to/from the network.
- For channels CAN2 and CAN3 only:  
The '5x 06' command can be used to force the AVT-423 to always send messages received from the CAN bus using only the '12 xx yy' long format.
- Example #1  
Turn on the time stamp function for CAN3.  
Command: 53 08 03 01.  
Header byte upper nibble 5 indicates a configuration command.  
Header byte lower nibble 3 indicates three bytes follow.  
\$08 is the time stamp command.  
\$03 indicates channel 3.  
\$01 enable time stamps.
- Example #2  
Send a message, to LIN0, as Master, ID = \$3C, 8 data bytes.  
Command: 0B 05 01 3C 01 02 03 04 05 06 07 08.  
Header byte = \$0B.  
Upper nibble \$0 indicates 'to the network'.  
Lower nibble \$B indicates 11 bytes follow.  
\$01 indicates send as Master.  
\$3C is the LIN message ID.  
\$01 02 03 ... are the 8 data bytes.
- Example #3  
Receive a message from CAN3, 11-bit ID, with 12 data bytes.  
Response: \$11 10 33 05 07 77 01 02 03 04 05 06 07 08 09 10 11 12  
Header byte: \$11, alternate header format #1.  
\$10: 16 bytes follow  
\$33: 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](http://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 Connections - Board Revisions “B”, “C”, and “D”**

<u>Pin #</u>	<u>Description</u>	<u>Notes</u>
1	CAN0_H	channel 0
14	CAN0_L	
2	CAN1_H	Transceiver <b>is</b> software selected. channel 1
15	CAN1_L	Transceiver <b>is</b> software selected.
3	CAN2_H	channel 2
16	CAN2_L	
4	CAN3_H	channel 3
17	CAN3_L	
5	FR-BUS_A_P	not available
18	FR-BUS_A_M	not available
6	FR-BUS_B_P	not available
19	FR_BUS_B_M	not available
7		
20		
8		
21		
9		
22		
10	CAN1_SWC	Transceiver software selected. channel 1
23	LIN0 or KWPO	LIN0 is channel 7 KWPO is channel 8
11	LIN1 or KWP1	LIN1 is channel 5

AVT-423 Multiple Interface

		KWP1 is channel 6
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 AVT-423 Connections - Board Revision "F"**

<u>Pin #</u>	<u>Description</u>	<u>Notes</u>
1	CAN0_H	channel 0
14	CAN0_L	
2	CAN1_H	Transceiver is software selected. channel 1
15	CAN1_L	Transceiver is software selected.
3	CAN2_H	channel 2
16	CAN2_L	
4	CAN3_H	channel 3
17	CAN3_L	
5	FR-BUS_A_P	not available
18	FR-BUS_A_M	not available
6	Digital Output # 1	Connected to Solid State Relay pin # 6
19	Digital Output # 1	Connected to Solid State Relay pin # 4
7		
20		
8		connected to JP1 pin # 3 LIN0 or KWP0 external supply <i>(board modification required)</i>
21		
9		connected to JP2 pin # 3 LIN1 or KWP1 external supply

## AVT-423 Multiple Interface

		<i>(board modification required)</i>
22		
10	CAN1_SWC	Transceiver software selected. channel 1
23	LIN0 or KWP0	LIN0 is channel 7 KWP0 is channel 8
11	LIN1 or KWP1	LIN1 is channel 5 KWP1 is channel 6
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 2

### 6.3 Power Requirements

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

#### 6.3.1 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.

#### 6.3.2 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.

#### 6.3.3 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

### 6.3.4 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.

## 8. DAC Connection

None at this time.

## 9. Digital Output

AVT-423 board revision “FF” is available on special request.  
It requires a factory modification to a production AVT-423 board.

The modification includes the addition of a Solid State Relay (SSR) to the top of the board.  
The SSR is controlled by the AVT-423 CPU by a Client command.

### 9.1 Digital Output Connection

The digital output is the switching side of the SSR.  
It is available on pins # 6 and # 19 of P3 – the DB25P connector on the AVT-423 board.

The SSR is configured such that the user can use it for switching AC or DC voltage and as either a high-side or low-side switch.

The SSR is: Vishay part number “VOR1121B6”.

A quick summary of SSR ratings:  
250 volt maximum  
200 mA maximum  
12 ohm on resistance, typical.

### 9.2 Digital Output Operation

The Client controls the digital output using the ‘5x 05’ command.

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

Refer to Section 17 for details about the ‘5x 05’ command.

## 10. 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.

### 10.1 Simultaneous Network Operations

With three 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.

LIN0 and KWP0 share a transceiver. Therefore, simultaneous operations for LIN0 and KWP0 are not possible. Pin # 23 on P3 can be operated as LIN0 (channel 7) or KWP0 (channel 8).

Exception #3.

LIN1 and KWP1 share a transceiver. Therefore, simultaneous operations for LIN1 and KWP1 are not possible. Pin # 11 on P3 can be operated as LIN1 (channel 5) or KWP1 (channel 6).

### 10.2 Start-up Parameters Stored in Non-Volatile Memory

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

The user can query for the stored value with the ‘51 80’ command.

The user can write desired stored values with the ‘55 80 rr ss tt vv’ command.

The parameters are stored as a bit map in a 32-bit long-word.

Unused bits are reserved and should be set to “1”.

At this time only bits 7:0 are used. They specify the default (power-on) states of the CAN transceivers and CAN termination – for CAN channels 0, 1, 2, and 3. The bits and states are:

bit7:	CAN3	0 = transceiver disabled (standby). 1 = transceiver enabled (operational).
bit6:	CAN2	0 = transceiver disabled (standby). 1 = transceiver enabled (operational).
bit5:	CAN1	0 = transceiver disabled (standby). 1 = transceiver enabled (operational).
bit4:	CAN0	0 = transceiver disabled (standby). 1 = transceiver enabled (operational).
bit3:	CAN3	0 = termination disabled (not in-circuit). 1 = termination enabled (in-circuit).
bit2:	CAN2	0 = termination disabled (not in-circuit). 1 = termination enabled (in-circuit).

bit1: CAN1      0 = termination disabled (not in-circuit).  
                  1 = termination enabled (in-circuit).  
bit0: CAN0      0 = termination disabled (not in-circuit).  
                  1 = termination enabled (in-circuit).

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

55 80 FF FF FF FD      Store the new setting.  
F1 A5                    Reset the AVT-423 application.

### **10.3 The Different ‘automatic’ responses; AVT-423 to Client**

Prior to firmware version 0034, the AVT-423 would send the same response to the Client when making an Ethernet connection and when conducting an ‘F1 A5’ reset.

This was discovered to be “not helpful” when debugging a problem.

As of firmware version 0036 the following actions result in the listed responses to be sent to the Client.

Power-on reset or a CPU reset.

91 0A.

Response to a successful Ethernet connection:

91 3A                    operational report.

93 04 xx yy            firmware version report.

Response to an ‘F1 A5’ reset command.

91 0F                    successful reset of the AVT-423 application.

### **10.4 Types of Resets**

The AVT-423 uses the “uCOS” operating system.

On power-up, the operating system then loads and runs the AVT-423 application.

#### **10.4.1 The ‘F1 A5’ Reset**

The ‘F1 A5’ reset only resets the AVT-423 application. It does not affect the Ethernet connection. Hence, that reset should not cause the Ethernet connection between the AVT-423 and Client to be lost.

This reset performs the following functions: It re-initializes all application variables and re-initializes all peripheral hardware; such as the CAN controllers and LIN UART controllers.

The response sent to the Client for the ‘F1 A5’ reset is: ‘91 0F’.

#### **10.4.2 The ‘F1 C3’ Reset**

The ‘F1 C3’ reset causes an operating system reset. This will cause loss of Ethernet connection, re-boot the operating system and then re-load the AVT-423 application. This is very close, but not quite the same as a power reset.

The response sent to the Client for the ‘F1 C3’ reset is: ‘91 0A’.

## 11. Network Hardware Descriptions

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

### 11.1 CAN0 - 2-wire CAN

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

It uses the Microchip MCP2562FD-E/SN transceiver.

CAN0 is not CAN-FD capable.

Refer to Tables 1 or 2 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.

#### 11.1.1 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.

### 11.2 CAN1 - 2-wire CAN or Single Wire CAN

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

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 Tables 1 or 2 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.

### **11.2.1 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.

### **11.3 CAN2 - 2-wire CAN**

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

It uses the Microchip MCP2562FD-E/SN transceiver.

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

Refer to Tables 1 or 2 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.

#### **11.3.1 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.

### **11.4 CAN3 - 2-wire CAN**

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

It uses the Microchip MCP2562FD-E/SN transceiver.

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

Refer to Tables 1 or 2 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.

#### **11.4.1 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.

### **11.5 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.

#### **11.5.1 LIN0 Channel Number**

LIN0 is designated channel 7.

### **11.6 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.

#### **11.6.1 LIN1 Channel Number**

LIN1 is designated channel 5.

### **11.7 KWP0**

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

Refer to Section 11.6, above, for information about the physical layer.

Refer to Tables 1 or 2 for pin number.

#### **11.7.1 KWP0 Channel Number**

KWP0 is designated channel 8.

### **11.8 KWP1**

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

Refer to Section 11.6, above, for information about the physical layer.

Refer to Tables 1 or 2 for pin number.

### **11.8.1 KWP1 Channel Number**

KWP1 is designated channel 6.

### **11.9 LIN2 thru LIN7**

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

Refer to Section 11.6, above, for information about the physical layer.

Refer to Tables 1 or 2 for pin number.

### **11.10 LIN2 thru LIN7**

LIN2 thru LIN7 are low speed, single wire, multi-drop, ground referenced networks.

These channels are only available when the AVT-424 LIN Expansion board is installed.

Refer to Appendix “A” for technical details about these additional LIN channels.

#### **11.10.1 LIN2 thru LIN7 Channel Numbers**

LIN2 thru LIN7 are designated by channels \$A thru \$F.

### **11.11 Flexray**

The hardware to implement Flexray was designed into the AVT-423 board.

However, Flexray capability will not be implemented.

The AVT-423 has Flexray hardware designed into the board. However, no firmware has been developed to support Flexray operations. I do NOT anticipate ever doing that firmware development.

The Flexray bus was to be implemented using the NXP (Freescale) MC9S12XF512MLM device.

A dual-bus Flexray interface was to be implemented.

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

#### **11.11.1 Flexray Channel Number**

Flexray was designated channel 9.

## **12. 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.

### **12.1 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.

## **12.2 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.

### **12.2.1 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.

### **12.2.2 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.

## **12.3 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.

### 12.3.1 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.

### 12.3.2 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.

### 12.3.3 Listen Only

This feature/function has not been implemented in firmware.

### 12.3.4 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 17.

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'.

#### 12.3.4.1 *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 17 for a complete description of all the transmit command formats.

#### 12.3.4.2 *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 17 for a complete description of all the transmit command formats.

#### 12.3.4.3 *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 17 for a complete description of all the transmit command formats.

#### **12.3.4.4 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.

#### **12.3.4.5 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 and inclusive).
- 12 data bytes (fixed).
- 16 data bytes (fixed).
- 20 data bytes (fixed).
- 24 data bytes (fixed).
- 32 data bytes (fixed).
- 48 data bytes (fixed).
- 64 data bytes (fixed).

#### **12.3.5 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 17.1.

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.

#### **12.3.5.1 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 17.1 for a description of all bytes in the packet.

#### **12.3.5.2 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 17.1 for a description of all bytes in the packet.

#### **12.3.5.3 Receive Response Format: \$12 xx yy**

The \$12 xx yy form of the receive response is used when the byte count of the response (not including the header byte) is \$FFFF or less. Refer to Section 17.1 for a description of all bytes in the packet.

#### **12.3.5.4 Transmit Echo**

For CAN2 and CAN3 only.

The Client can use the '5x 40' command to force the AVT-423 to echo all transmitted CAN frames.

The time stamp of the echo (if enabled) is as of when the CAN frame is received by the CAN controller that transmitted it.

Only CAN frames transmitted by a transmit command or from a frame buffer can be echoed. CAN frames transmitted by a periodic message or ISO 15765 transaction can NOT be echoed.

#### **12.3.5.5 Long Response Only**

For CAN2 and CAN3 only.

The Client can use the '5x 06' command to select the receive response format the AVT-423 will use to send received CAN messages to the Client.

When 'long form only' is enabled, the AVT-423 will always use the '12 xx yy' format, regardless of the actual byte count of the response.

#### **12.3.5.6 Receive Response: Object number**

As of version '0043', for CAN2 and CAN3, the object number is low nibble only of the actual object that received the message.

### **12.3.6 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.

#### **12.3.6.1 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.

#### ***12.3.6.2 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.

#### ***12.3.6.3 Transmit Acknowledgment Description***

Refer to Section 17.1 for a complete description of the transmit ack response to the Client with and without time stamps.

#### ***12.3.6.4 Transmit Acknowledgement– Object number***

As of version '0043', for CAN2 and CAN3, the object number is low nibble only of the actual object that transmitted the message.

### ***12.4 Object ID and Mask***

#### **12.4.1 Configuration – CAN0 and CAN1**

Each CAN channel is independent of all other channels.

Each channel has 16 (decimal) message objects numbered: \$0 to \$F.

Each message object can be configured as either transmit or receive.

Each message object, when configured for receive, can be set for 11 or 29-bit IDs.

Each message object, when configured for receive, has an associated mask.

Each bit of the mask can be set for “must match” or “don't care”. The default is all bits are “must match”. A '1' in a bit position means “must match”.

The combination of the object ID and associated mask give the user flexibility as to what messages are received by the designated object.

#### **12.4.2 Configuration – CAN2 and CAN3**

Each CAN channel is independent of all other channels.

Each channel has 64 (decimal) receive message objects.

Each channel has 64 (decimal) transmit message objects.

The message objects are numbered: \$00 to \$3F.

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

### 12.4.3 Object ID and Mask Operation

Object IDs and Masks are related.

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

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

Using the '\$76 2A ...' form of the object command specifies an 11-bit ID.  
(for objects numbered 0x00 to 0x3F. CAN2 and CAN3 only..)

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

Using the '\$78 2A ...' form of the object command specifies a 29-bit ID.  
(for objects numbered 0x00 to 0x3F. CAN2 and CAN3 only..)

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.

### 12.4.4 Acceptance ID and Mask Notes

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

### 12.4.5 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.

### 12.4.6 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 2A' command 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.

#### 12.4.6.1 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 # 4
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.)

### 12.5 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.

### 12.5.1 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

```

### 12.6 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.

#### 12.6.1 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
```

### 12.6.2 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 2C 12 34 56 78

; set CAN2 maskC to all must match.
77 2C 02 2C 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
```

### 12.6.3 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.

### **12.7 Periodic Message Support**

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

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

#### **12.7.1 Periodic Message Number and Numbering**

Channels CAN0, CAN1, CAN2, and CAN3 each have 32 (decimal) periodic messages available.

The periodic messages are numbered: 0x00 to 0x1F.

#### **12.7.2 Periodic Message Data Field Length**

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

For CAN0 and CAN1 – only the '7x 18' command is used to define a periodic message.

Channels CAN2 and CAN3 support CAN-FD periodic messages with data fields of 0 to 64 bytes.

#### **12.7.3 Long Periodic Message Definition and Response**

The following only applies to channels CAN2 and CAN3.

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

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

Refer to Section 17, immediately following the '7x 18' command definition, for the 'long form' command.

Likewise, refer to Section 17.1, immediately following the '8x 18' response definition, for the 'long form' response.

A 'long format' command will return a 'long form' response.

A 'long format' query will return a 'long form' response.

A '7x 18' command will return an '8x 18' response.

A '7x 18' query will return an '8x 18' response if the data field is less than or equal to 8 bytes.

A '7x 18' query will return a 'long form' response if the data field is more than 8 bytes.

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

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

#### 12.7.4 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.

##### 12.7.4.1 Type1 Example

We 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.

### **12.7.5 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).

### **12.7.6 Periodic Message Commands**

All commands are listed in Section 17. 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.

### **12.8 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.

#### **12.8.1 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).

### **12.8.2 Special Function: RC6**

Implemented for channels CAN2 and CAN3 only.

The so-called "RC6" function is available for CAN periodic messages.

A brief description of the RC6 function follows. This only applies to a periodic message for which this function has been enabled.

A 4-bit rolling counter is updated. The counter is stored in the lower nibble of Data1 of the CAN frame data field. Bytes Data1 to the end of the data field are then run through a CRC computation algorithm: Each data byte is used as an index into a table named "CRC8H2FTable". The resulting byte is then 'XOR'ed with the running result. The 4-bit counter value is used as an index into a user specified table to obtain a byte. That byte is 'XOR'ed with the previous running result. That result is then used as an index into "CRC8H2FTable" to obtain the final CRC value. The final CRC value is inserted into Data0 of the CAN frame.

At that point the CAN frame is queued for transmission and the periodic timer is reset.

### **12.8.3 Special Function xxx**

xxx

### **12.8.4 Special Function xxx**

xxx

## **12.9 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.

### **12.9.1 Periodic Pause Function Command**

This function has not yet been implemented in firmware.

Refer to the \$7x 1F command in Section 17 for detailed information regarding the command format.

## **12.10 Frame Buffers**

This Section only applies to channels CAN2 and CAN3.

With firmware version '0056' the Frame Buffer capability was introduced.

### **12.10.1 Frame Buffer Description**

There are 16 (decimal) frame buffers assigned to each channel: CAN2 and CAN3 only.

The frame buffers are numbered: 0x00 to 0x0F (inclusive).

Each frame buffer consists of the following:

- A fixed CAN arbitration ID (11 or 29-bit).
- A timer (msec resolution).
- A set of 64 (decimal) ‘slots’.
  - Each slot holds the data field of a CAN message.
  - Maximum slot size is 64 (decimal) bytes.

Each frame buffer can be configured to operate in either of two modes:

- FIFO mode.
- Rolling mode.

### **12.10.2 Frame Buffer FIFO Mode**

The Client sets up and enables the CAN channel for operations.  
(At a minimum: set baud rate and enable the channel.)

The Client sets the frame buffer mode to FIFO.

The ‘7x 52’ command.

The Client sets the frame buffer time interval between transmissions.

The ‘7x 51’ command.

The Client sets the frame buffer arbitration ID.

The ‘7x 50’ command.

The Client enables the frame buffer.

The ‘7x 54’ command.

Each time the Client writes a data field for transmission, that data is stored in the next available ‘slot’.

Setting bit 6 of the channel byte in an ‘11 xx’ or ‘12 xx yy’ long form transmit command indicates a frame buffer write. Refer to Sections 5.3 and 17 for command format details.

The Client can write as many data fields as it wants, as fast as it wants.

Limited only by number of available empty slots.

The writes are stored in the slots, in a FIFO configuration.

The AVT-423 performs the following:

- When the frame buffer timer expires, the oldest (first written) slot is read.
- The data from that slot is combined with the arbitration ID, and the EDL and BRS bits, to create a complete CAN frame.
- That CAN frame is loaded into the appropriate CAN channel controller for immediate transmission to the CAN bus.
- The frame buffer timer is reset.

### 12.10.3 Frame Buffer FIFO Mode Notes

If the FIFO goes empty, no transmission occurs and the timer is reset. This will continue until a slot is written.

If the frame buffer (slot) write command is valid, the data is stored and no response is returned to the Client.

If the frame buffer (slot FIFO) is full – the write command will be ignored and a '31 12' command error response will be returned.

CAN frames with no data are accepted.

### 12.10.4 Frame Buffer Rolling Mode

The Client sets up and enables the CAN channel for operations.  
(At a minimum: set baud rate and enable the channel.)

The Client sets the frame buffer mode to Rolling.  
The '7x 52' command.

The Client sets the frame buffer time interval between transmissions.  
The '7x 51' command.

The Client sets the frame buffer arbitration ID.  
The '7x 50' command.

The Client enables the frame buffer.  
The '7x 54' command.

Each time the Client writes a data field for transmission, that data is stored in the next available 'slot'.

Setting bit 6 of the channel byte in an '11 xx' or '12 xx yy' long form transmit command indicates a frame buffer write. Refer to Sections 5.3 and 17 for command format details.

The Client can write as many data fields as it wants, as fast as it wants.

The writes are stored in the slots, in the order written.

The AVT-423 performs the following:

- When the frame buffer timer expires, the 'next' slot is read.  
(The 'slot read pointer' goes from top to bottom and then resets to the top.)
- The data from that slot is combined with the arbitration ID, and the EDL and BRS bits, to create a complete CAN frame.
- That CAN frame is then loaded into the appropriate CAN channel controller for immediate transmission to the CAN bus.

### 12.10.5 Frame Buffer Rolling Mode Notes

If the slot write command is valid, the data is stored and no response is returned to the Client.

If the number of writes exceed the number of slots, subsequent writes over-write the oldest data.  
(The 'slot write pointer' goes from top to bottom and then resets to the top.)

### 12.10.6 Frame Buffer General Notes

CAN frames with no data are accepted.

A 'no data' frame is **not** the same as a 'no transmit' slot.

The Client can write a 'no transmit' command to a slot.

A 'no transmit' slot causes the AVT-423 to read that slot, does **not** load anything for transmission, but it does reset the frame buffer timer.

Only valid EDL and BRS bit combinations are accepted.

For a write command with an invalid EDL and BRS bit combination – the write command will be ignored and a '31 12' command error response will be returned.

The EDL bit must be set for a write command with more than 8 data bytes.

If not, a '31 12' error response will be returned.

For CAN-FD data fields, the AVT-423 does **not** check the length of the data field in the write command to ensure it is a valid CAN-FD length.

Valid CAN-FD data lengths are (decimal):

- 0 to 8 (inclusive)
- 12
- 16
- 20
- 24
- 32
- 48
- 64

For example: If the Client sends a write command with 34 bytes of data, the AVT-423 will load a CAN-FD frame of 48 data bytes. The last 14 bytes will be of indeterminate value. Therefore, if data field padding is necessary, the Client must do it.

Frame buffer parameters, arb ID, and time interval can be changed at any time (on-the-fly).

The Client should disable the frame buffer and flush the frame buffer – before switching modes.

### 12.10.7 Frame Buffer Command Summary

Commands:

- '7x 50' – Frame buffer arb ID.
- '7x 51' – Frame buffer transmit interval time.
- '7x 52' – Frame buffer mode.
- '7x 53' – Query for frame buffer slot status and clear the frame buffer.
- '7x 54' – Frame buffer status.

The frame buffer write command form:

- 11 xx 4r qs aa bb cc ...
  - or
  - 12 xx yy 4r qs aa bb cc ...
- (either form is acceptable)
- xx: count of bytes to follow
  - xx yy: count of bytes to follow

4: indicates a frame buffer write  
r: can channel 2, 3  
q: b7: 0 (not used)  
b6: 0 = normal write, with or without data  
1 = a 'no transmit' slot write  
b5: EDL bit: 0 = classical can  
1 = can-fd  
b4: BRS bit: 0 = data field at normal speed  
1 = data field at high speed  
s: frame buffer number, 0x00 to 0x0F  
aa bb cc ... : data field; 0 to 64 bytes.

The frame buffer 'no transmit' write command form:

11 02 4r 4s

or

12 00 02 4r 4s

(either form is acceptable)

r: can channel 2, 3

s: frame buffer number, 0x00 to 0x0F

Note: The command can be longer.

The EDL and BRS bits are ignored.

The data field is ignored.

### **12.11 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 12.12 for information about ISO 15765 support for channels CAN2 and CAN3.

#### **12.11.1 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.

### **12.11.2 Differences with ISO 15765 on CAN0/1 (as compared to CAN2/3)**

Channels CAN0 and CAN1 each have a total of 16 objects (numbered \$0 to \$F).

An object can be configured for receive or transmit. Only one configuration per object.

Channels CAN2 and CAN3 each have 64 transmit and 64 receive objects. (Technically, a total of 128 objects.) Note that a receive object can ONLY be configured for receive operations. Similarly for transmit objects.

All objects are configured using the '7x 2A' command and enabled using the '7x 04' command.

### **12.11.3 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.

#### ***12.11.3.1 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 17 for complete and detailed description of each command.

- Set CAN0 to 500 Kbaud.  
74 0A 00 02
- Object #2 ID = \$246.  
75 2A 00 02 02 46
- Enable object #2 for transmit.  
74 04 00 02 02
- Object #3 ID = 0357.  
75 2A 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.

### ***12.11.3.2 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).

### ***12.11.3.3 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.

## ***12.12 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 12.12 for information about ISO 15765 support for channels CAN0 and CAN1.

### **12.12.1 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 64 receive objects (numbered 0x00 to 0x3F) and 64 transmit objects (numbered 0x00 to 0x3F).

(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.

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

Channels CAN2 and CAN3 each have 64 transmit objects (numbered 0x00 to 0x3F).

Channels CAN2 and CAN3 each have 64 receive objects (numbered 0x00 to 0x3F).

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

Receive objects are configured using the '7x 2A' command and enabled using the '7x 04' command.

### 12.12.3 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.

#### 12.12.3.1 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 17 for complete and detailed description of each command.

- Set CAN2 to 500 Kbaud.  
73 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 2A 02 08 02 46
- Object #8 acceptance mask, all bits must match.  
75 2C 02 28 07 FF  
(EDL bit is 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.

### ***12.12.3.2 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).

### ***12.12.3.3 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.

#### **12.12.4 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.

##### **12.12.4.1 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 17 for complete and detailed description of each command.

- Set CAN2 to 500 Kbaud and 2 Mbaud.  
74 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 2A 02 08 02 46
- Object #8 acceptance mask, all bits must match.  
75 2C 02 28 07 FF  
(EDL bit in the object byte is 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.

#### **12.12.4.2 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).

#### **12.12.4.3 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.

### **12.13 CAN ‘Mirror’ Function**

This function and the related commands are new as of firmware version 0032.

#### **12.13.1 Operational Description**

The ‘mirror’ function exists for each CAN channel, separately. Cross channels operations are not supported.

For a specific CAN channel, ...

- The user specifies a receive ID.
- The user specifies the transmit ID and object number.
- The user enables the function.

The AVT watches all CAN frames received from the network (regardless of object number). If a received ID matches the specified receive ID – the AVT-423 holds the entire data field (including BRS and EDL bits, if applicable). The AVT-423 then immediately composes and queues for transmission using the specified object number, a CAN frame using the specified transmit ID and the held data field. The data field is not changed.

The turn-around time (end of the receive frame to start of the transmit frame) is ~140 usec +/- 20 usec.

#### **12.13.2 Commands**

Three new commands were introduced to support the ‘mirror’ function. Refer to Section 17 for command details.

The ‘7x 20’ command is used to specify the CAN channel number and receive ID.

The '7x 21' command is used to specify the CAN channel number, transmit object number, and transmit ID.

The '7x 22' command is used to disable or enable the 'mirror' function for a specified CAN channel number.

### **13. LIN1 Operations**

LIN1 is channel number 5.

LIN1 operation is independent of all other channels.

LIN1 operation is controlled by the \$53 69 05 0y command.

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

LIN1 hardware is shared with KWP1 operations. Only one mode can be enabled at a time. Either LIN1 or KWP1, but not both.

All of the following LIN discussions apply to both LIN channels, LIN1 and LIN0 (channel numbers 5 and 7, respectively).

#### **13.1 Communications**

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

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

#### **13.2 LIN1 Bus Supply Voltage**

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

The supply for the AVT-423 LIN1 transceiver and that for the connected module is usually the same supply. If they are not the same, then they should be 'close' in voltage level to one another.

Both the AVT-423 and the connected module should share the same ground.

On the AVT-423 board, the LIN1 bus pull-up resistor is 1 K ohm.

On AVT-423 boards revisions 'D' and below, the LIN1 bus supply is the same as the supply that powers the AVT-423 board.

On AVT-423 boards revision 'F' and above, the default configuration is the LIN1 bus pull-up supply is the same as the supply to the AVT-423 board. However, the board design allows the user to select a separate external LIN1 bus supply (that is NOT the supply to the AVT-423 board).

#### **13.3 LIN1 Bus Supply – Selecting a Separate External Supply**

To change the configuration of the AVT-423 to allow use of a separate external supply for the LIN1 transceiver, the following modification need to be completed.

Note: This can ONLY be done for boards of revision 'F' and above.

- On the top of the AVT-423 PC board, locate three small holes labeled “JP2”. JP2 is located near fuse F1 and sort-of between F1 and P3 (the DB-25P connector).
- Very carefully cut the trace between JP2 pins # 1 and 2.
- Install a small jumper wire between JP2 pins # 2 and 3.  
Or, install a three pin header for JP2 and install a shunt across pins # 2 and 3.
- The external supply for LIN1 transceiver is now through P3 (the DB-25P connector) pin # 9.

To change it back to the original configuration, remove any connection between JP2 pins # 2 and 3. Then connect JP2 pins # 1 and 2.

### 13.3.1 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.

### 13.3.2 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.

### 13.3.3 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-423 will throw out an ID byte only message and not tell the Client.

The \$53 66 command selects whether or not the AVT-423 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.

```

### 13.3.4 Communications Example

This example enables LIN1 operations, 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 LIN1 network message (passively)
05 05 00 C4 78 9A
;    0 indicates from the network
;    5 count of bytes to follow
;    05 channel 5 – LIN1
;    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 – LIN1
;    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 – LIN1
;    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 – LIN1
;    00 slave node
;    C4 message ID
;    11 22 message data

```

### 13.3.5 Time Stamp

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

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

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).

#### 13.3.5.1 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 – LIN1
    00 status byte indicating no errors detected.
    25 message ID.
    11 22 33 44 message bytes.
```

When time stamps are enabled a receive message example is:

```
0C rr ss tt vv 05 00 25 11 22 33 44
    0A header byte, indicates from the network, $A or decimal 10 bytes follow.
    rr ss tt vv time stamp.
    05 channel 5 – LIN1
    00 status byte indicating no errors detected.
    25 message ID.
    11 22 33 44 message bytes.
```

#### 13.3.5.2 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 – LIN1
    40 status byte, bit 5 set, indicates from this node.
```

When time stamps are enabled a transmit ack example is:

```
06 rr ss tt vv 05 60
    04 header byte, indicates from the network, 4 bytes follow.
    rr ss tt vv time stamp (xx is the high byte, yy is the low byte).
    05 channel 5 – LIN1
    40 status byte, bit 5 set, indicates from this node.
```

### 13.4 Special Functions

This section describes all special functions available for LIN.

### 13.4.1 LIN Frame Data Definition

Each LIN frame can contain up to 8 data bytes.

In the following discussions, Data0 is the first data byte in the LIN frame.

Likewise Data7 is the last byte of the LIN frame.

Within a byte, the bits are numbered from 0 (least significant bit) to 7 (most significant).

### 13.4.2 LIN Mirror Function

The LIN mirror function permits the user to specify a receive and a transmit ID. When set-up and enabled, the AVT-423 will ‘watch’ all frames on the LIN bus. If a frame is received with ID that matches the specified receive ID, the data from that frame is then transmitted by the AVT-423 using the specified ID. The ‘mirror’ message is transmitted as either a Master or Slave; as specified by the user in the set-up command. A new checksum is appended using the LIN checksum method; as set by the ‘5x 5A’ command.

The user specifies how the ‘mirror’ message is to be transmitted. If ‘Master’ mode is specified, the ‘mirror’ message is queued for transmission and is transmitted as soon as possible. If ‘Slave’ mode is specified, the ‘mirror’ message is queued for transmission. The AVT-423 waits for the matching ID to appear on the LIN bus. If the expected ID is not received within 100 msec, the queued ‘mirror’ message is discarded.

#### 13.4.2.1 LIN Mirror Command Summary

There are two commands associated with the LIN mirror function. They are the ‘set-up’ and the ‘enable’ commands. Both commands are described here. Command definitions are in Section 17 and responses are in Section 17.1.

52 11 0y: LIN mirror function set-up query.

y: LIN channel, 5 or 7.

55 11 0y rr 0m tt: LIN mirror function set-up.

y: LIN channel, 5 or 7.

rr: receive ID to watch for.

m: 0 = transmit as slave.

1 = transmit as master.

tt: transmit ID.

52 12 0y: LIN mirror function status query.

y: LIN channel, 5 or 7.

53 12 0y 0s: LIN mirror function status command.

y: LIN channel, 5 or 7.

s: 0 = disable.

1 = enable.

### 13.4.3 Special Function 1

Does not currently exist.

### **13.5 Periodic Message Support**

When LIN 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).

#### **13.5.1 LIN Frame Data Definition**

Each LIN frame can contain up to 8 data bytes.

In the following discussions, Data0 is the first data byte in the LIN frame.

Likewise Data7 is the last byte of the LIN frame.

Within a byte, the bits are numbered from 0 (least significant bit) to 7 (most significant).

#### **13.5.2 Modes of Operation**

LIN periodic messages are defined as either Master or Slave messages – as specified in the 7x 18 periodic message set-up command.

A periodic message designated as Master will only operate as Type1. Type1 periodic messages are transmitted independent of one another.

Type2 periodic message are not implemented.

A periodic message designated as Slave operate as described in Section 13.5.5, below.

#### **13.5.3 Organization of Periodic Messages**

There are sixteen (decimal) periodic messages. The periodic messages are numbered: \$0 to \$F (inclusive).

Each message is independently configured (\$7x 18 05 command)

Each message is independently disabled or enabled (\$7x 1B 05 command).

Each message has its own time interval (\$7x 1A 05 command). The time interval is 1 msec.

#### **13.5.4 Type1 Periodic Message**

Type1 periodic messages operate independently of each other.

When Type1 operations are enabled, each enabled message in that group operates according to its own interval count.

- The message is set up.

- The interval count is defined.

- The message is enabled.

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

### 13.5.4.1 Type1 Example

Here is a sequence of commands to set-up and enable a Master periodic message.

Remember: 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.

### 13.5.5 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 LIN bus, all periodic messages are searched. If a periodic message is a slave and enabled, and if its ID byte matches that just received from the LIN bus, then that message is immediately transmitted into the data field of the LIN frame in progress.

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

### 13.5.6 Periodic Message Commands

All commands are listed in Section 17 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.

### 13.6 Periodic Message Special Functions

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

#### 13.6.1 LIN Frame Data Definition

Each LIN frame can contain up to 8 data bytes.

In the following discussions, Data0 is the first data byte in the LIN frame. Likewise Data7 is the last byte of the LIN frame.

Within a byte, the bits are numbered from 0 (least significant bit) to 7 (most significant).

### **13.6.2 Periodic Message Special Function 0**

Does not currently exist.

#### **13.6.2.1 PM-SF0 Command**

Does not currently exist.

### **13.7 ABIC Support**

LIN1 does NOT support communications with an ABIC module.

### **13.8 Commands and Responses**

Refer to Section 17 for a complete list of LIN commands and Section 17.1 for responses.

## **14. LIN0 operations**

LIN0 is channel number 7.

LIN0 operation is independent of all other channels.

LIN0 operation is controlled by the \$53 69 07 0z command.

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

LIN0 hardware is shared with KWPO operations. Only one mode can be enabled at a time. Either LIN0 or KWPO, but not both.

### **14.1 LIN0 Operation Notes**

LIN0 operations are nearly identical to the LIN operations described in Section 13, above.

### **14.2 LIN0 Bus Supply Voltage**

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

The supply for the AVT-423 LIN0 transceiver and that for the connected module is usually the same supply. If they are not the same, then they should be 'close' in voltage level to one another.

Both the AVT-423 and the connected module should share the same ground.

On the AVT-423 board, the LIN0 bus pull-up resistor is 1 K ohm.

On AVT-423 boards revisions 'D' and below, the LIN0 bus supply is the same as the supply that powers the AVT-423 board.

On AVT-423 boards revision 'F' and above, the default configuration is the LIN0 bus pull-up supply is the same as the supply to the AVT-423 board. However, the board design allows the user to select a separate external LIN0 bus supply (that is NOT the supply to the AVT-423 board).

### **14.3 LIN0 Bus Supply – Selecting a Separate External Supply**

To change the configuration of the AVT-423 to allow use of a separate external supply for the LIN0 transceiver, the following modification need to be completed.

Note: This can ONLY be done for boards of revision ‘F’ and above.

- On the top of the AVT-423 PC board, locate three small holes labeled “JP1”. JP1 is located near fuse F1 and sort-of between F1 and P3 (the DB-25P connector).
- Very carefully cut the trace between JP1 pins # 1 and 2.
- Install a small jumper wire between JP1 pins # 2 and 3.  
Or, install a three pin header for JP1 and install a shunt across pins # 2 and 3.
- The external supply for LIN0 transceiver is now through P3 (the DB-25P connector) pin # 8.

To change it back to the original configuration, remove any connection between JP1 pins # 2 and 3. Then connect JP1 pins # 1 and 2.

### **14.4 Commands and Responses**

Refer to Section 17 for a complete list of LIN commands and Section 17.1 for responses.

## **15. KWP1 operations**

KWP1 is channel 6.

KWP1 operation is independent of all other channels.

KWP1 operation is controlled by the \$53 69 06 0z command.

KWP1 hardware is shared with LIN1 operations. Only one mode can be enabled at a time. Either KWP1 or LIN1, but not both.

### **15.1 KWP1 Operation Limitations**

As of version ‘0049’ the following KWP operations are supported:

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

### **15.2 KWP1 Bus Supply Voltage**

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

The supply for the AVT-423 KWP1 transceiver and that for the connected module is usually the same supply. If they are not the same, then they should be ‘close’ in voltage level to one another.

Both the AVT-423 and the connected module should share the same ground.

On the AVT-423 board, the KWP1 K-line pull-up resistor is 1 K ohm.

On AVT-423 boards revisions 'D' and below, the KWP1 K-line supply is the same as the supply that powers the AVT-423 board.

On AVT-423 boards revision 'F' and above, the default configuration is the KWP1 K-line pull-up supply is the same as the supply to the AVT-423 board. However, the board design allows the user to select a separate external KWP1 K-line supply (that is NOT the supply to the AVT-423 board).

### **15.3 KWP1 K-line Supply – Selecting a Separate External Supply**

To change the configuration of the AVT-423 to allow use of a separate external supply for the KWP1 transceiver, the following modification need to be completed.

Note: This can ONLY be done for boards of revision 'F' and above.

- On the top of the AVT-423 PC board, locate three small holes labeled "JP2". JP2 is located near fuse F1 and sort-of between F1 and P3 (the DB-25P connector).
- Very carefully cut the trace between JP2 pins # 1 and 2.
- Install a small jumper wire between JP2 pins # 2 and 3.  
Or, install a three pin header for JP2 and install a shunt across pins # 2 and 3.
- The external supply for KWP1 transceiver is now through P3 (the DB-25P connector) pin # 9.

To change it back to the original configuration, remove any connection between JP2 pins # 2 and 3. Then connect JP2 pins # 1 and 2.

### **15.4 Commands and Responses**

Refer to Section 17 for a complete list of KWP commands and Section 17.1 for responses.

### **15.5 KWP1 Operation – Quick Intro**

A "quick start" guide to KWP operations follows.

Connect the external K-line to pin # 11.

Remember that the AVT-423 and the module must have a common ground and the power supplies must be close in voltage level.

Select KWP1 to be active. Note that this will force LIN1 to be disabled.

53 69 06 01

Set the K-line baud rate.

53 50 06 0x

At this point any message that shows up on the K-line will be passed to the Client.

Other commands that you may want to use.

Select checksum type: 5x 4B.

Transmit inter-byte time: 5x 27.

Enable/Disable format byte processing: 5x 28.

Receive buffer time-out (if format byte processing is disabled): 5x 02.

### 15.6 KWP1 Operation – Fast Init

KWP1 operations support ‘Fast Init’ as defined in ISO14230.

A brief description of how ‘Fast Init’ operations.

- Fast Init is invoked using the ‘6x 13 0y rr ss ...’ command.

Notes:

‘x’ is the count of bytes to follow.

‘y’ is the channel number (8 – for KWP1).

‘rr ss ...’ is the start communications message that the connected module is expecting.

A common example is: 65 13 08 81 11 81.

‘6’ – an init command.

‘5’ – five bytes follow.

‘13’ – fast init.

‘8’ – channel 8, KWP1.

‘81’ – physical addressing, one data byte.

‘11’ – physical address of module.

‘81’ – start communications service.

the checksum is appended by the AVT-423 (unless that function is disabled).

- Fast init sequence does not proceed until the K-line has been observed to be idle for time ‘W5’.  
‘W5’ default is 300 msec.  
related ‘W5’ command: 5x 46.
- The AVT-423 will wait for 600 msec for the K-line to be idle for ‘W5’ before declaring a failure and responding with:  
‘2x yy’ (transmit watchdog expired)  
and  
‘73 13 08 01’ (fast init failure).
- The rest of this sequence follows only after the K-line is idle for ‘W5’.
- The K-line is pulled low and held low for 25 msec.  
25 msec is the default setting.  
related command: 5x 47.
- The K-line is released to go high for 25 msec.  
25 msec is the default setting.  
related command: 5x 48.
- Transmission of the start communications message begins.  
The communications default baud rate is 10400.  
related command: 5x 50.
- After the start communications message has completed without error, the AVT-423 will respond with ‘73 13 08 11’ (fast init success).
- During the fast init sequence, most error conditions will be detected. Any error will cause the fast init sequence to terminate and the Client will receive an error response from the AVT-423.

## 16. KWP0 operations

KWP0 is channel 8.

KWP0 operation is independent of all other channels.

KWP0 operation is controlled by the \$53 69 08 0z command.

KWP0 hardware is shared with LIN0 operations. Only one mode can be enabled at a time. Either KWP0 or LIN0, but not both.

### 16.1 KWP0 Operation Limitations

As of version '0049' the following KWP operations are supported:

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

### 16.2 KWP0 Bus Supply Voltage

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

The supply for the AVT-423 KWP0 transceiver and that for the connected module is usually the same supply. If they are not the same, then they should be 'close' in voltage level to one another.

Both the AVT-423 and the connected module should share the same ground.

On the AVT-423 board, the KWP0 K-line pull-up resistor is 1 K ohm.

On AVT-423 boards revisions 'D' and below, the KWP0 K-line supply is the same as the supply that powers the AVT-423 board.

On AVT-423 boards revision 'F' and above, the default configuration is the KWP0 K-line pull-up supply is the same as the supply to the AVT-423 board. However, the board design allows the user to select a separate external KWP0 K-line supply (that is NOT the supply to the AVT-423 board).

### 16.3 KWP0 K-line Supply – Selecting a Separate External Supply

To change the configuration of the AVT-423 to allow use of a separate external supply for the KWP0 transceiver, the following modification need to be completed.

Note: This can ONLY be done for boards of revision 'F' and above.

- On the top of the AVT-423 PC board, locate three small holes labeled "JP1". JP1 is located near fuse F1 and sort-of between F1 and P3 (the DB-25P connector).
- Very carefully cut the trace between JP1 pins # 1 and 2.
- Install a small jumper wire between JP1 pins # 2 and 3.  
Or, install a three pin header for JP1 and install a shunt across pins # 2 and 3.

- The external supply for KWP0 transceiver is now through P3 (the DB-25P connector) pin # 8.

To change it back to the original configuration, remove any connection between JP1 pins # 2 and 3. Then connect JP1 pins # 1 and 2.

#### **16.4 Commands and Responses**

Refer to Section 17 for a complete list of KWP commands and Section 17.1 for responses.

#### **16.5 KWP0 Operations**

All operations and commands for KWP0 are identical to that for KWP1. Therefore, refer to the KWP1 section, above.

## 17. 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 this order (left to right).

```
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].

**CAN2, CAN3**

0x 1r q0 ss tt vv ww zz mm nn ... :

x: count of bytes to follow.  
 1: extended object number support  
 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.  
 ss: object number: 0x00 to 0x3F.  
 tt vv: 11-bit ID, right justified.  
 tt vv ww zz: 29-bit ID, right justified.  
 mm nn ... : data [optional].

**LIN0, LIN1, and LIN2 thru LIN7**

0x 0z 0m id aa bb cc ... :

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

**KWP0, KWP1**

0x 0z aa bb cc ... :

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

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**

11 xx 1r q0 ss tt vv ww zz mm nn ... :

xx: count of bytes to follow.  
 1: extended object number support  
 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.  
 ss: object number: 0x00 to 0x3F.  
 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.

**CAN2, CAN3**

12 xx yy 1r q0 ss tt vv ww zz mm nn ... :  
 xx yy: count of bytes to follow.  
 1: extended object number support  
 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.  
 ss: object number: 0x00 to 0x3F.  
 tt vv: 11-bit ID, right justified.  
 tt vv ww zz: 29-bit ID, right justified.  
 mm nn ... : data.

**CAN2, CAN3 Frame Buffer Write Command**

11 xx 4r qs aa bb cc ...  
 or  
 12 xx yy 4r qs aa bb cc ...  
 (either form is acceptable)  
 xx: count of bytes to follow  
 xx yy: count of bytes to follow  
 4: indicates a frame buffer write  
 r: can channel 2, 3  
 q: b7: 0 (not used)  
     b6: 0 = normal write, with or without data  
         1 = a 'no transmit' slot write  
     b5: EDL bit: 0 = classical can  
                   1 = can-fd

b4: BRS bit: 0 = data field at normal speed  
                   1 = data field at high speed  
 s: frame buffer number, 0x00 to 0x0F  
 aa bb cc ... : data field; 0 to 64 bytes.

**CAN2, CAN3 Long Periodic Message support**

Refer to the '7x 18' command for the long periodic message definition command.  
 The form is '11 bb 2r ...' or '12 00 bb 2r ...'.

Data byte count limitations CAN2 and CAN3

Maximum is 64 bytes with EDL bit = 1 and non-ISO 15765 operations.  
 Refer to Section 12.3.4.5 for list of valid data field sizes.  
 Maximum is \$2000 (8192 decimal) for ISO 15765 enabled object.

**KWP0, KWP1**

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

**KWP0, KWP1**

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

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: Hardware reset of AVT-424. This does NOT generate a response.  
 21 31: Firmware reset of AVT-424.

3: \_\_\_\_\_

4: \_\_\_\_\_

5: Configuration.

---

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

52 01 0r: Send received checksum to Client, status query.  
 r: channel: 7, 5, 8, 6, and A thru F.

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

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

52 02 0r: Receive buffer timeout query.  
 r: channel: 7, 5, 8, 6, and A thru F.

53 02 0r zz: Set receiver buffer timeout.  
 r: channel: 7, 5, 8, 6, and A thru F.  
 zz: time in milliseconds.  
 [Default = 3 msec. for all LIN channels.]  
 [Default = 25 msec. (\$19) for all KWP channels.]

-----  
**Not channel specific**

51 05: Query for digital output status.

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

-----  
**CAN2, CAN3**

52 06 0r: Query for status of response format for received CAN messages.  
 r: channel: 2 or 3.

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

-----  
**CAN0, CAN1, CAN2, CAN3, LIN0, LIN1, KWP0, KWP1, and LIN2 thru LIN7**

52 08 0y: Time stamp status query.  
 y: channel: 0, 1, 2, 3, 7, 5, 8, 6, and A thru F.

53 08 0y 0s: Disable / Enable time stamp.

---

---

y: channel: 0, 1, 2, 3, 7, 5, 8, 6, and A thru F.  
s: 0: Disable. [Default]  
1: Enable – uses 1 msec timer.  
2: Enable – uses native timer.  
(CAN0 and CAN1: baud clock.)  
(CAN2 and CAN3: 2 kHz clock.)

-----  
**LINO, LIN1**

52 11 0y: LIN mirror function set-up query.  
y: LIN channel: 7, 5.

55 11 0y rr 0m tt: LIN mirror function set-up.  
y: LIN channel: 7, 5.  
rr: receive ID to watch for.  
m: 0 = transmit as slave.  
1 = transmit as master.  
tt: transmit ID.

-----  
**LINO, LIN1**

52 12 0y: LIN mirror function status query.  
y: LIN channel: 7, 5.

53 12 0y 0s: LIN mirror function status command.  
y: LIN channel: 7, 5.  
s: 0 = disable. [Default]  
1 = enable.

-----  
**LINO, LIN1**

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

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

-----  
**KWPO, KWP1**

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

54 27 0r zz: Set P4. (P4 is transmit inter-byte time.)  
r: channel: 8, 6.  
zz: time in milliseconds.

---

[Default = 5 msec.]

-----  
**KWP0, KWP1**

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

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

-----  
**KWP0, KWP1**

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

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

-----  
**Not channel specific**  
**(do not use this command)**  
**(for debug and engineering development use only)**

52 31 0y:     getsockopt(fd).  
           y:     socket number: 1.

54 31 0y 01 00:   send:            clrsockopt(fd, SO\_NONAGLE)  
 54 31 0y 01 01:   send:            setsockopt(fd, SO\_NONAGLE)  
 54 31 0y 02 00:   send:            clrsockopt(fd, SO\_NOPUSH)  
 54 31 0y 02 01:   send:            setsockopt(fd, SO\_NOPUSH)  
 54 31 0y 03 xx    send:            SetSocketTxBuffers(fd, xx)  
 54 31 0y 04 xx    send:            setsocketackbuffers(fd, xx)  
 54 31 0y 05 xx    send:            SetSocketRxBuffers(fd, xx)  
 54 31 0y 06 xx    send:            SetOutOfOrderbuffers(fd, xx)  
           y:     socket number: 1.

-----  
**CAN0, CAN1, CAN2, CAN3, LIN0, LIN1, KWP0, KWP1, and LIN2 thru LIN7**

52 40 0r:     Transmit ack status query.  
           r:     channel: 0, 1, 2, 3, 7, 5, 8, 6, and A thru F.

53 40 0r 0y:   Disable / Enable transmit acks.  
           r:     channel: 0, 1, 2, 3, 7, 5, 8, 6, and A thru F.

---

y: 0 disable.  
 1 enable. [Default]  
 2 echo enable.  
 (option 2 only valid for channels CAN2 and CAN3.)

-----  
**KWP0, KWP1**

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

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

-----  
**KWP0, KWP1**

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

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

-----  
**KWP0, KWP1**

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

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

-----  
**KWP0, KWP1**

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

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

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

52 50 0r: Baud rate query.  
 r: channel: 7, 5, 8, 6, and A thru F.

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

**AVT-423 only**

54 50 0r yy zz: Set baud rate.  
 r: channel: 7, 5, 8, 6.  
 yy zz: divisor load.  
 Baud rate formula. (Numbers shown are decimal.)  

$$\text{Baud rate} = 125,000,000 / 32 / \text{yy zz}$$
 ('yy zz' is converted to decimal)

**AVT-424 only**

54 50 0r yy zz: Set baud rate.  
 r: channel: A thru F.  
 yy zz: divisor load.  
 Baud rate formula. (Numbers shown are decimal.)  

$$\text{Baud rate} = 1,000,000 / \text{yy zz}$$
 ('yy zz' is converted to decimal)

-----  
**LIN0, LIN1, and LIN2 thru LIN7**

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

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

-----  
**LIN0, LIN1, and LIN2 thru LIN7**

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

53 5A 0r 0z: Set checksum type.  
 r: LIN channel: 7, 5, and A thru F.  
 z: 0: classic (LIN 1.3).

- 1: enhanced (LIN 2.0). [Default]
- 2: none.

-----  
**LIN0, LIN1, and LIN2 thru LIN7**

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

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

**KWP0, KWP1**

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

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

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

52 69 0r: Secondary operations status query.  
 r: channel numbers: 7, 5, 8, 6, and A thru F.

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

Notes:

LIN0 and KWP0 operations are mutually exclusive – they share hardware.

LIN1 and KWP1 operations are mutually exclusive – they share hardware.

-----  
**AVT-423 only**

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.]

-----  
**AVT-424 only**

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

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

-----  
**LIN0, LIN1, KWPO, KWP1, and LIN2 thru LIN7**

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

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

-----  
**Not channel specific**

51 80: Query for stored (non-volatile) start-up parameters.  
 55 80 rr ss tt vv: Set stored (non-volatile) start-up parameters.  
 rr ss tt vv is the following bit map.

[default state of every bit is '1']

- b31: '1' reserved, not used.
- b30: '1' reserved, not used.
- b29: '1' reserved, not used.
- b28: '1' reserved, not used.
- b27: '1' reserved, not used.
- b26: '1' reserved, not used.
- b25: '1' reserved, not used.
- b24: '1' reserved, not used.
- b23: '1' reserved, not used.
- b22: '1' reserved, not used.
- b21: '1' reserved, not used.
- b20: '1' reserved, not used.
- b19: '1' reserved, not used.
- b18: '1' reserved, not used.
- b17: '1' reserved, not used.
- b16: '1' reserved, not used.
- b15: '1' reserved, not used.
- b14: '1' reserved, not used.
- b13: '1' reserved, not used.
- b12: '1' reserved, not used.
- b11: '1' reserved, not used.
- b10: '1' reserved, not used.
- b09: '1' reserved, not used.
- b08: '1' reserved, not used.
- b07: CAN3 transceiver status

0 = standby / disabled.  
 1 = operate / enabled.



r: channel: 2, 3.  
 z: 0 = ISO operations. [Default]  
 1 = Bosch operations.

-----  
**CAN0, CAN1**

73 04 0r 0y: Object status query.  
 r: channel: 0, 1.  
 y: object number \$0 to \$F.

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

**CAN2, CAN3**

73 04 0r yy: Object status query.  
 r: channel: 2, 3.  
 yy: object number: 0x00 to 0x3F.

74 04 0r yy 0z: Disable / Enable object.  
 r: channel: 2, 3.  
 yy: object number: 0x00 to 0x3F.  
 z: 0: object disabled. [Default]  
 1: object enabled for receive.

-----  
**CAN0, CAN1**

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

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

7x 06 0r 0z rr ss tt ...: Set object data.  
 r: channel: 0, 1.  
 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.

z: object number: \$0 to \$F.

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

-----  
**CAN2, CAN3**  
**(do not use this command)**  
**(for debug and engineering development use only)**

71 09: Query for maximum count of received frames processed before exiting the CAN receive manager.

72 09 yy: Set maximum count of received frames processed before exiting the CAN receive manager.  
 yy: count. [Default: \$02].

-----  
**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]

-----  
**CAN0, CAN1, CAN2, CAN3**

**(this command only valid if hardware has been modified)**

72 13 0r: Query for CAN transceiver status.  
r: channel: 0, 1, 2, 3.

73 13 0r 0s: Set CAN transceiver status.  
r: channel: 0, 1, 2, 3.  
s: 0 = disable 2-wire transceiver (standby).  
1 = enable 2-wire transceiver (operate). [Default]

-----  
**CAN2, CAN3 (Transmit object only.)**

73 17 0r zz: Query for transmit object configuration.  
r: CAN channel: 2, 3.  
zz: object number: 0x00 to 0x3F.

A command will elicit a response of the same form.  
A query will elicit a response of the appropriate form.

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.  
 [Default: 00 00]

76 17 0r w0 zz 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)  
 zz: object number: 0x00 to 0x3F.  
 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.

78 17 0r w0 zz 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)  
 zz: object number: 0x00 to 0x3F.  
 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 bytes 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 (optional; 0 to 8 bytes).  
 [Default: 00 00]

**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 (optional; 0 to 8 bytes).  
 [Default: 00 00]

**CAN2, CAN3**

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

Long Form Query

11 03 2r 00 pp

12 00 03 2r 00 pp

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

Long Form Command

11 bb 2r y0 pp tt vv ww zz mm nn ...

12 00 bb 2r y0 pp tt vv ww zz mm nn ...

- 11: long header.
- 12 00: long header.
- bb: count of bytes to follow.
- 2: long periodic message command.
- r: channel number: 2, 3.
- y:
  - b7: IDE 1: 29-bit.
  - 0: 11-bit.
  - b6: RTR 1: is an RTR frame.
  - 0: is NOT an RTR frame.
  - b5: EDL 1: CAN-FD.
  - 0: Classical CAN.
  - b5: BRS 1: high speed data field.
  - 0: normal speed for entire frame.
- 0
- pp: message number: \$00 to \$2F.
- tt vv: 11-bit id.
- tt vv ww zz: 29-bit id.
- mm nn ... : data field (0 to 64 bytes, inclusive).  
Refer to Section 12.3.4.5 for valid data field lengths.

**LIN0, LIN1, and LIN2 thru LIN7**

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

7x 18 0r pp 0m rr ss tt ... : Periodic message set-up.  
r: channel: 7, 5, and A thru F.  
pp: message number: \$0 to \$F.  
m: 0: slave.

1: master.  
 rr ss tt ... : data field. [optional]

**KWPO, KWP1**

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

7x 18 0x 0p rr ss tt ... : Periodic message set-up.  
 x: channel number: 8, 6.  
 p: message number: \$0 to \$F.  
 rr ss tt ... : data field.

-----  
**CAN0, CAN1**

73 19 0r pp: Query for object assignment for CAN periodic message.  
 r: channel: 0, 1.  
 pp: message number: \$00 to \$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.  
 [Default: 0]

-----  
**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, KWPO, KWP1, and LIN2 thru LIN7**

73 1A 0r zz: Periodic message disable/enable status query.  
 r: channel: 7, 5, 8, 6, and A thru F.  
 zz: message number: \$0 to \$F.

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

-----  
**CAN0, CAN1, CAN2, CAN3**

73 1B 0r zz: Periodic message interval count status query.  
 r: channel: 0, 1, 2, 3.  
 zz: message number: \$00 to \$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, KWP0, KWP1, and LIN2 thru LIN7**

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

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

-----  
**CAN0, CAN1, CAN2, CAN3, LIN0, LIN1, KWP0, KWP1, and LIN2 thru LIN7**

72 1C 0r: Disable all periodic message for channel 'r'.  
 r: channel: 0, 1, 2, 3, 7, 5, 8, 6, and A thru F.  
 r: 'FF' – disable all messages, all channels.

-----  
**CAN2, CAN3**

72 1F 0r: Query for state of processing received CAN frames.  
 r: channel: 2, 3.

73 1F 0r 0s: Control processing of received CAN frames.  
 r: channel: 2, 3.  
 s: 0 = do not process received CAN frames (discard them).  
 1 = process received CAN frames. [Default]

-----  
**CAN0, CAN1, CAN2, CAN3**

72 20 0r: Query for 'mirror' function receive ID.  
 r: channel: 0, 1, 2, 3.

75 20 0r 00 0s tt: Set 'mirror' function receive 11-bit ID.  
 r: channel: 0, 1, 2, 3.

00.  
 st: 11-bit ID.  
 [Default: 00 00]

77 20 0r 00 ss tt vv ww: Set 'mirror' function receive 29-bit ID.  
 r: channel: 0, 1, 2, 3.  
 00.  
 ss tt vv ww: 29-bit ID.

-----  
**CAN0, CAN1, CAN2, CAN3**

72 21 0r: Query for 'mirror' function transmit ID.  
 r: channel: 0, 1, 2, 3.

75 21 0r 0p 0s tt: Set 'mirror' function transmit 11-bit ID.  
 r: channel: 0, 1, 2, 3.  
 p: object number: \$0 to \$F.  
 stt: 11-bit ID.  
 [Default: 00 00]

77 21 0r 0p ss tt vv ww: Set 'mirror' function transmit 29-bit ID.  
 r: channel: 0, 1, 2, 3.  
 p: object number: \$0 to \$F.  
 ss tt vv ww: 29-bit ID.

-----  
**CAN0, CAN1, CAN2, CAN3**

72 22 0r: Query for 'mirror' function status.  
 r: channel: 0, 1, 2, 3.

73 22 0r 0s: Set 'mirror' function status.  
 r: channel: 0, 1, 2, 3.  
 s: 0 = disable. [Default]  
 1 = enable.

-----  
**CAN0, CAN1, CAN2, CAN3**

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

73 25 0r ss: Set the flow control additional separation time.  
 r: channel: 0, 1, 2, 3.  
 ss: additional time, in milliseconds,  
 to add to the responding node separation time.  
 [Default: 00 00]

**CAN0, CAN1**

- 73 27 0r 0z: Query for ISO 15765 padding status.  
 r: channel: 0, 1, 2, 3.  
 z: object number: \$0 to \$F.
- 74 27 0r 0z 0v: Disable / enable ISO 15765 padding status.  
 r: channel: 0, 1, 2, 3.  
 z: object number: \$0 to \$F.  
 v: 0: disable.  
 1: enable. [Default]
- 75 27 0r 0z 0v ww: Disable / enable ISO 15765 padding status.  
 r: channel: 0, 1, 2, 3.  
 z: object number: \$0 to \$F.  
 v: 0: disable.  
 1: enable. [Default]  
 ww: pad byte. [Default = \$FF.]

**CAN2, CAN3**

- 73 27 0r zz: Query for ISO 15765 padding status  
 r: channel: 0, 1, 2, 3.  
 zz: object number: 0x00 to 0x3F.
- 74 27 0r zz 0v: Disable / enable ISO 15765 padding status.  
 r: channel: 0, 1, 2, 3.  
 zz: object number: 0x00 to 0x3F.  
 v: 0: disable.  
 1: enable. [Default]
- 75 27 0r zz 0v ww: Disable / enable ISO 15765 padding status.  
 r: channel: 0, 1, 2, 3.  
 zz: object number: 0x00 to 0x3F.  
 v: 0: disable.  
 1: enable. [Default]  
 ww: pad byte. [Default = \$FF.]

-----  
**CAN0, CAN1**

- 72 28 0r: Query for pairing status, all objects.  
 r: channel: 0, 1, 2, 3.
- 73 28 0r 0y: Disable object pairing, object 'y' and its mate.  
 Disable ISO 15765 operations.  
 r: channel: 0, 1, 2, 3.  
 y: object number: \$0 to \$F.
- 74 28 0r 0y 0s: Pair the two objects for ISO 15765 operations.

r: channel: 0, 1, 2, 3.  
 y: object number: \$0 to \$F.  
 s: object number: \$0 to \$F.

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

**CAN2, CAN3**

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

73 28 0r yy: Disable object pairing, object 'y' and its mate.  
 Disable ISO 15765 operations.  
 r: channel: 0, 1, 2, 3.  
 yy: object number: 0x00 to 0x3F.

74 28 0r yy ss: Pair the two objects for ISO 15765 operations.  
 r: channel: 0, 1, 2, 3.  
 yy: object number: 0x00 to 0x3F.  
 ss: object number: 0x00 to 0x3F.

75 28 0r yy ss ww: Pair the two objects for ISO 15765 operations and set 'AE' byte.  
 r: channel: 0, 1, 2, 3.  
 yy: object number: 0x00 to 0x3F.  
 ss: object number: 0x00 to 0x3F.  
 ww: set 'AE' byte.

-----  
**CAN2, CAN3**

73 29 0r yy: Query for 'max\_dlc'.  
 r: channel: 2, 3.  
 yy: object number: 0x00 to 0x3F.

74 29 0r yy ss: Set 'max\_dlc'. (Only used with ISO15765 processing.)  
 r: channel: 2, 3.  
 yy: object number: 0x00 to 0x3F.  
 ss: only the following values are valid (hex digits):  
 08, 0C, 10, 14, 18, 20, 30, 40.  
 [Default: \$40 = 64]

-----  
**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 zz: Report object configuration.  
 r: channel: 2, 3.  
 zz: object number: 0x00 to 0x3F.

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.

76 2A 0r y0 zz 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.  
 zz: object number: 0x00 to 0x3F.  
 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.

78 2A 0r y0 zz 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.  
 zz: object number: 0x00 to 0x3F.  
 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 = \$1FFFFFF.]

**CAN2, CAN3**

73 2C 0r zz: Report mask.  
 r: channel: 2, 3.  
 zz: mask number: 0x00 to 0x3F.  
  
 1: Bit must match.  
 0: Bit is don't care.

75 2C 0r yz ss tt: Specify 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.  
 [Default = \$7FF.]

76 2C 0r y0 zz ss tt: Specify 11-bit mask.  
 r: channel: 2, 3.  
 y: b7: IDE bit.  
 b6: 0.  
 b5: EDL bit.  
 b4: 0.  
 zz: mask number: 0x00 to 0x3F.  
 ss tt: mask value, 11-bit.  
 [Default = \$7FF.]

77 2C 0r yz ss tt vv ww: Specify 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.  
 [Default = \$1FFFFFF.]

78 2C 0r y0 zz ss tt vv ww: Specify 29-bit mask.  
 r: channel: 2, 3.  
 y: b7: IDE bit.  
 b6: 0.

b5: EDL bit.  
 b4: 0.  
 zz: mask number: 0x00 to 0x3F.  
 ss tt vv ww: mask value, 29-bit.  
 [Default = \$1FFFFFF.]

-----  
**CAN0, CAN1**

73 30 0r 0z: Query for ISO 15765 'AE' status and byte.  
 r: channel: 0, 1.  
 z: object number: \$0 to \$F.

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

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

**CAN2, CAN3**

73 30 0r zz: Query for ISO 15765 'AE' status and byte.  
 r: channel: 0, 1, 2, 3.  
 zz: object number: 0x00 to 0x3F.

74 30 0r zz 0s: Disable / Enable ISO 15765 'AE' operation.  
 r: channel: 0, 1, 2, 3.  
 zz: object number: 0x00 to 0x3F.  
 s: 0 = disable. [Default]  
 1 = enable.

75 30 0r zz 0s ww: Disable / Enable ISO 15765 'AE' operation; specify 'AE' byte.  
 r: channel: 0, 1, 2, 3.  
 zz: object number: 0x00 to 0x3F.  
 s: 0 = disable. [Default]  
 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**

73 50 0y 0z: Query for frame buffer arb ID.  
 y: channel: 2, 3.  
 z: frame buffer number: 0x00 to 0x0F.

75 50 0y 0z 0r ss: Set frame buffer 11-bit arb ID.  
 y: channel: 2, 3.  
 z: frame buffer number: 0x00 to 0x0F.  
 r ss: 11-bit arb ID.  
 [Default: \$07FF]

77 50 0y 0z rr ss tt vv: Set frame buffer 29-bit arb ID.  
 y: channel: 2, 3.  
 z: frame buffer number: 0x00 to 0x0F.  
 rr ss tt vv: 29-bit arb ID.

-----  
**CAN2, CAN3**

73 51 0y 0z: Query for frame buffer timer reset value.  
 y: channel: 2, 3.  
 z: frame buffer number: 0x00 to 0x0F.

75 51 0y 0z rr ss: Set frame buffer timer reset value.  
 y: channel: 2, 3.  
 z: frame buffer number: 0x00 to 0x0F.  
 rr ss: timer reset value in msec. [Default: \$03E8 = 1000 msec.]

-----  
**CAN2, CAN3**

73 52 0y 0z: Query for frame buffer mode.  
 y: channel: 2, 3.  
 z: frame buffer number: 0x00 to 0x0F.

74 52 0y 0z 0r: Set frame buffer mode.  
 y: channel: 2, 3.  
 z: frame buffer number: 0x00 to 0x0F.  
 r: 0 = FIFO mode. [Default]  
 1 = Rolling mode.

-----  
**CAN2, CAN3**

73 53 0y 0z: Query for frame buffer slot status.  
 y: channel: 2, 3.

---

	z:	frame buffer number: 0x00 to 0x0F.
74 53 0y 0z EE:		Clear and disable the frame buffer.
	y:	channel: 2, 3.
	z:	frame buffer number: 0x00 to 0x0F.
-----		
<b>CAN2, CAN3</b>		
73 54 0y 0z:		Query for frame buffer status.
	y:	channel: 2, 3.
	z:	frame buffer number: 0x00 to 0x0F.
74 54 0y 0z 0r:		Set frame buffer status.
	y:	channel: 2, 3.
	z:	frame buffer number: 0x00 to 0x0F.
	r:	0 = disable. [Default] 1 = enable.
-----		
<b>CAN2, CAN3</b>		
73 5F 0r ss:		Query for RC6 status.
	r:	channel: 2, 3.
	ss:	periodic message number (0x00 to 0x1F).
74 5F 0r ss 0z:		Set RC6 status.
	r:	channel: 2, 3.
	ss:	periodic message number (0x00 to 0x1F).
	y:	0 = disabled. [Default] 1 = enabled, use KA_01. 2 = enabled, use Klemmen_Status_01. 3 = enabled, use Motor_Code_01. 4 = enabled, use UserStateServer_01. 5 = enabled, use 'user defined'. (not implemented, yet)
-----		
<b>CAN2, CAN3</b>		
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.
-----		
<b>CAN2, CAN3</b>		
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 = 400]

8: \_\_\_\_\_

9: \_\_\_\_\_

A: \_\_\_\_\_

B: Firmware version.

- B0: Request firmware version number.  
(Deprecated. Do not use this command.) (Use the 'B1 01' command.)
- B1 01: Request firmware version number.
- B1 02: Request CAN-FD IP core version number.
- B1 03: Request model number.
- B1 04: Request MAC address.
- B1 05: Request mezzanine board (AVT-424) firmware version number.
- B1 06: Request mezzanine board model number.

C: \_\_\_\_\_

D: Reserved.  
debug commands. do not use.

E: xxx.

F: Model Query and Reset

- F0: Query for model number.  
(Deprecated. Do not use this command.) (Use the 'B1 03' command.)
- F1 A5: Restart the AVT-423 (a software reset).  
This also resets the mezzanine board (AVT-424).
- F1 C3: Reset the Netburner CPU.  
(This will result in an Ethernet disconnect.)

**17.1 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 Az: Transmit ack.  
 r: channel number: 2, 3.  
 z: transmit object number, low nibble only.

**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 Az: Transmit ack.  
 jj kk ll mm: time stamp  
 r: channel number: 2, 3.  
 z: transmit object number, low nibble only.

**LIN0, LIN1, KWP0, KWP1**

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

**LIN0, LIN1, KWP0, KWP1**

06 jj kk ll mm 0r pp: Transmit ack.  
 jj kk ll mm: time stamp  
 r: channel number: 7, 5, 8, 6.  
 pp: receive status byte (defined below).

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; low nibble only.  
 tt vv: 11-bit ID, right justified.  
 tt vv ww zz: 29-bit ID, right justified.  
 nn pp... : data.

**LIN0, LIN1, and LIN2 thru LIN7**

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

LIN status byte

b07: buffer closed by frame time out.  
 b06: from this device.  
 b05: from my periodic message (AVT-424).  
 b04: buffer closed by last byte timer.  
 b03: buffer opened without break.  
 b02: buffer closed due to max byte count.  
 b01: buffer closed by break.  
 b00: checksum error.

**KWP0, KWP1**

0x jj kk ll mm Or ss tt vv ww ... :

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

KWP status byte

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

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

-----  
**CAN0, CAN1**

11 xx jj kk ll mm Or 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: \$0 to \$F; low nibble only.

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 1r q0 ss tt vv ww zz nn pp... :

xx: count of bytes to follow.

jj kk ll mm: time stamp [optional].

1: extended object support.

---

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.  
 ss: object number: 0x00 to 0x3F.  
 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: \$0 to \$F; low nibble only.  
   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 1r q0 ss tt vv ww zz nn pp... :  
   xx yy: count of bytes to follow.  
   jj kk ll mm: time stamp [optional].  
   1: extended object support.  
   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.
ss:	object number:	0x00 to 0x3F.
tt vv:	11-bit ID,	right justified.
tt vv ww zz:	29-bit ID,	right justified.
nn pp ... :	data.	

**CAN2, CAN3**

Refer to the '8x 18' response for the long periodic message definition response. The form is '11 bb 3r ...'.

2: Error Responses.

-----  
21 01: Inbound command too long, flushed.

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

-----  
22 03 01: FIFO1p1 overflow.

22 03 02: FIFO1p2 overflow.

22 03 03: FIFO1p3 overflow.

22 03 04: FIFO1p4 overflow.

-----  
22 04 xx: Bad FIFO42\_state variable.  
xx: byte read from FIFO4.

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

-----  
21 06:

-----  
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.

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

-----  
2x 0E:        Error in return code when setting TCP parameters.  
              Temporary. That code is currently commented out.  
              Refer to "tcp\_010.cpp".

-----  
2x 0F:

-----  
21 10:        CAN0 initialization error.

-----  
21 11:        CAN1 initialization error.  
-----

---

21 12:        CAN2 initialization error.

-----

21 13:        CAN3 initialization error.

-----

2x 14:

-----

2x 15:

-----

2x 16:

-----

2x 17:

-----

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

-----

21 19:        FIFO3 full after transfer of command.  
              in: "send\_cmd\_to\_424".

-----

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

-----

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

-----

22 1C ss:     UART2 error flags, bit map  
              b7:     buffer1 was used  
              b6:  
              b5:  
              b4:

---

---

b3:  
b2:  
b1:  
b0: no buffer available, byte discarded

-----  
2x 1D:

-----  
2x 1E:

-----  
2x 1F:

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

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

-----  
22 22 0y: Channel index error in kwp\_rcv\_mgr.  
y: channel number.

-----  
23 23 0x 0y: Buffer index error in kwp\_rcv\_mgr.  
x: channel index.  
y: buffer index.

-----  
23 24 0x yy: One byte message in kwp\_rcv\_mgr.  
x: channel number.  
yy: the one byte.

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

-----  
24 26 0x yy zz: UART errors in kwp0\_new\_byte.  
x: channel number.  
yy: receive status.  
zz: receive data.

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

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

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

-----  
23 2A xx yy: Invalid channel index in kwp\_xmt\_mgr.  
xx yy: channel index.

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

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

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

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

-----  
22 2F 0x: Invalid transmit buffer state.  
x: channel number.

-----  
22 30 0x: Transmit buffer watchdog expired.  
x: channel number.

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

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

-----  
23 33 0x yy: 6x 13 command error, transmit state no idle  
x: channel number.  
yy: transmit state.

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

-----  
23 35 0x yy: 6x 13 command error, transmit buffer state not idle.  
x: channel number.  
yy: buffer state.

-----  
22 36 0x: Fast Init manager, invalid channel index.  
x: channel index.

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

-----  
23 38 0x yy: Fast Init manager, invalid state.  
x: channel number.  
yy: transmit state.

-----  
2x 39:

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

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

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

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

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

-----  
22 3F rr: Invalid channel number in 'kwp\_lf\_mgr'.  
rr: channel number.

-----  
22 40 0x: LIN\_mirror\_xmt\_mgr, channel index error, channel 'x'.

-----  
23 41 01 0x: Channel index error, UART1, channel 'x'.  
23 41 02 0x: Channel index error, UART2, channel 'x'.

-----  
22 42 0x: Transmit index error, xmt\_ix 'x'.

-----  
22 43 0x: Buffer index error, buff\_ix 'x'.

-----  
2x 44:

-----  
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.

-----  
 2x 50:

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

-----  
 25 52 0x yy rr ss: CAN channel x, receive warning  
                   x: channel number  
                   yy: receive counter  
                   rr ss: error status register (low word) (ERRSTAT)

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

---

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

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

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

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

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

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

-----  
2x 58:

-----  
2x 59:

-----  
2x 5A:

-----  
2x 5B:

-----  
2x 5C:

-----  
2x 5D:

-----  
2x 5E:

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

-----  
22 5F 01:     canA\_xmt\_12: data count too long, 11-bit id, 'ae' disabled.

-----  
22 5F 02:     canA\_xmt\_12: data count too long, 29-bit id, 'ae' disabled.

-----  
22 5F 03:     canA\_xmt\_12: data count too long, 11-bit id, 'ae' enabled.

-----  
22 5F 04:     canA\_xmt\_12: data count too long, 29-bit id, 'ae' enabled.

-----  
22 5F 05:     canA\_iso\_xmt\_buff\_mgr: buff\_state = 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.

-----  
23 5F 43 0y: canB\_iso\_rcv\_mgr, DLC > 8 and EDL not set.  
'y' is CAN channel number.

-----  
23 5F 44 0y: canB\_iso\_rcv\_mgr, DLC too short, with ae.  
'y' is CAN channel number.

-----  
23 5F 45 0y: canB\_iso\_rcv\_mgr, DLC too short, without ae.  
'y' is CAN channel number.

-----  
23 5F 46 0y: canB\_iso\_rcv\_mgr, (frame\_cnt < pci\_cnt).  
          'y' is CAN channel number.

-----  
23 5F 47 xx: canB\_iso\_rcv\_mgr, invalid buffer number for consecutive frame.  
          'xx' = buffer number.

-----  
23 5F 48 0y: canB\_iso\_rcv\_mgr, invalid buffer state.  
          'y' is CAN channel number.

-----  
23 5F 49 0y: canB\_iso\_rcv\_mgr, invalid consecutive frame sequence number.  
          'y' is CAN channel number.

-----  
23 5F 4A 0y: canB\_iso\_rcv\_mgr, frame\_cnt = zero in first frame.  
          'y' is CAN channel number.

-----  
23 5F 4B 0y: canB\_iso\_rcv\_mgr, pci\_cnt = 0 in first frame.  
          'y' is CAN channel number.

-----  
23 5F 4C 0y: canB\_iso\_rcv\_mgr, pci\_cnt > 8192 in first frame.  
          'y' is CAN channel number.

-----  
23 5F 4D 0y: canB\_iso\_rcv\_mgr, no buffer available.  
          'y' is CAN channel number.

-----  
23 5F 4E 0y: canB\_iso\_rcv\_mgr, flow control frame too short.  
          'y' is CAN channel number.

-----  
23 5F 4F 0y: canB\_iso\_rcv\_mgr, buffer not expecting flow control frame.  
          'y' is CAN channel number.

-----  
23 5F 50 0y: canB\_iso\_rcv\_mgr, invalid separation time in flow control frame, \$80 to \$F0.  
          'y' is CAN channel number.  
-----

---

23 5F 51 0y: canB\_iso\_rcv\_mgr, invalid separation time in flow control frame, > \$FA.  
Set to 1 msec.  
'y' is CAN channel number.

-----  
23 5F 52 0y: canB\_iso\_rcv\_mgr, invalid flow status in flow control frame.  
'y' is CAN channel number.

-----  
23 5F 53 0y: canB\_iso\_rcv\_mgr, unknown frame type.  
'y' is CAN channel number.

-----  
22 5F 54: canB\_iso\_rcv\_buff\_mgr, invalid channel number.

-----  
22 5F 55: canB\_iso\_rcv\_buff\_mgr, invalid buffer number.

-----  
23 5F 56 0y: canB\_iso\_rcv\_buff\_mgr, buffer time-out, first frame received,  
flow control transmit pending.  
'y' is CAN channel number.

-----  
23 5F 57 0y: canB\_iso\_rcv\_buff\_mgr, buffer time-out while receiving data.  
'y' is CAN channel number.

-----  
23 5F 58 0y: canB\_iso\_rcv\_buff\_mgr, time-out waiting to send buffer to Client.  
'y' is CAN channel number.

-----  
22 5F 59: canB\_xmt\_0x, RTR can not be true for ISO 15765 frame.

-----  
22 5F 5A: canB\_xmt\_12, RTR can not be true for ISO frame.

-----  
22 5F 5B: canB\_iso\_xmt\_proc, 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.

-----  
23 5F 68 0y: canB\_iso\_xmt\_buff\_mgr, buffer state = 12, watchdog expired.  
'y' is CAN channel number.

-----  
23 5F 69 0y: canB\_iso\_xmt\_buff\_mgr, buffer state = 13, watchdog expired.  
'y' is CAN channel number.

-----  
22 5F 6A: canB\_iso\_xmt\_buff\_mgr, buffer state = 13, byte count = 0.

-----  
23 5F 6B 0y: canB\_iso\_xmt\_buff\_mgr, buffer state = 13 buffer time-out.  
'y' is CAN channel number.

-----

---

23 5F 6C 0y: canB\_iso\_xmt\_buff\_mgr, buffer state = 13, buffer time-out.  
'y' is CAN channel number.

-----  
22 5F 6D 0y:

-----  
22 5F 6E: canB\_iso\_buff\_mgr, invalid channel number.

-----  
22 5F 6F: canB\_iso\_buff\_mgr, invalid buffer number.

-----  
23 5F 70 0y: canB\_iso\_buff\_mgr, invalid buffer state.  
'y' is CAN channel number.

-----  
23 5F 71 0y: canB\_iso\_buff\_mgr, invalid buffer state.  
'y' is CAN channel number.

-----  
23 5F 72 0y: canB\_iso\_buff\_mgr, invalid object number.  
'y' is CAN channel number.

-----  
23 5F 73 0y: canB\_iso\_buff\_mgr, invalid mate entry.  
'y' is CAN channel number.

-----  
22 5F 74:

-----  
22 5F 75:

-----  
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.

-----  
23 5F 81 0y:  canB\_iso\_xmt\_buff\_mgr, bad dlc decode, buffer state = 13, no ae.  
              'y' is CAN channel number.

-----  
22 5F 82:     canB\_iso\_xmt\_buff\_mgr, bad dlc decode, buffer state = 13, with ae.

-----  
23 5F 83 0y:  canB\_iso\_buff\_mgr: invalid obj\_buff entry.  
              'y' is CAN channel number.

-----  
22 5F 84:

-----  
2x 60 00:

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

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

-----  
23 60 03 yy:  AVT-424     Command not processed.

yy: header byte of offending command.

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

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

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

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

-----  
 2x 61:

-----  
 Notes:

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

All 'ss ...' codes are listed below.

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

- b15:
- b14:
- b13:
- b12:
  
- b11:
- b10: buffer1 invalid state > 0x06
- b09: buffer1 invalid state
- b08: buffer0 invalid state > 0x06

b07: buffer0 invalid state == 0  
 b06: rcvd byte, not break, no buffer available  
 b05: rcvd byte, not break, no active buffer open  
 b04: break byte not 00  
  
 b03: buff1 synch byte error  
 b02: buff0 synch byte error  
 b01: break rcvd, buff0 and buff1 not idle  
 b00: k-line short to ground detected

-----  
 24 6r 02 ss tt:      UART error(s) detected with received byte  
           r:            LIN channel number 2 thru 7  
           ss tt:        bit map follows  
           b15:    UTXISEL1 - xmt interrupt mode bit1  
           b14:    UTXINV - xmt polarity  
           b13:    UTXISEL0 - xmt interrupt mode bit0  
           b12:    URXEN - rcv enable  
  
           b11:    UTXBRK - xmt a break symbol  
           b10:    UTXEN - xmt enable  
           b09:    UTXBF - xmt buffer full  
           b08:    TRMT - xmt shift register empty  
  
           b07:    URXISEL1 - rcv interrupt mode bit1  
           b06:    URXISEL0 - rcv interrupt mode bit0  
           b05:    ADDEN - address character detect  
           b04:    RIDLE - rcv idle  
  
           b03:    PERR - parity error  
           b02:    FERR - framing error  
           b01:    OERR - rcv buffer overrun  
           b00:    URXDA - rcv data available

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

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

-----  
 22 6r 05:            'xmt\_type' error in transmit command processor.

---

r:	LIN channel number 2 thru 7
-----	
22 6r 06:	Transmit command too short, 12 xx yy.
r:	LIN channel number 2 thru 7
-----	
22 6r 07:	Transmit command too long, 12 xx yy, (do not add checksum).
r:	LIN channel number 2 thru 7
-----	
22 6r 08:	Transmit command too long, 12 xx yy, (do add checksum).
r:	LIN channel number 2 thru 7
-----	
22 6r 09:	Transmit command too short, slave.
r:	LIN channel number 2 thru 7
-----	
22 6r 0A:	Transmit command too short, master.
r:	LIN channel number 2 thru 7
-----	
22 6r 0B:	Transmit command too long, 0x, (do add checksum).
r:	LIN channel number 2 thru 7
-----	
22 6r 0C:	Transmit command too long, 0x, (do not add checksum).
r:	LIN channel number 2 thru 7
-----	
22 6r 0D:	Transmit buffer state == 0; invalid.
r:	LIN channel number 2 thru 7
-----	
22 6r 0E:	Transmit buffer state > 0x03; invalid.
r:	LIN channel number 2 thru 7
-----	
22 6r 0F:	Transmit buffer watchdog expired.
r:	LIN channel number 2 thru 7
-----	
22 6r 10:	not defined
r:	LIN channel number 2 thru 7

---

---

----- 22 6r 11: r:	Received byte not equal to xmt byte; data byte. LIN channel number 2 thru 7
----- 22 6r 12: r:	not defined LIN channel number 2 thru 7
----- 22 6r 13: r:	not defined LIN channel number 2 thru 7
----- 22 6r 14: r:	not defined LIN channel number 2 thru 7
----- 22 6r 15: r:	Received byte not equal to xmt byte; synch byte. LIN channel number 2 thru 7
----- 22 6r 16: r:	Received byte not equal to xmt byte; id byte only transmission. LIN channel number 2 thru 7
----- 22 6r 17: r:	Received byte not equal to xmt byte; last data byte. LIN channel number 2 thru 7
----- 22 6r 18: r:	Invalid transmit state in lin2_xmt_mgr. LIN channel number 2 thru 7
----- 22 6r 19: r:	Invalid transmit state in lin2_xmt_mgr. LIN channel number 2 thru 7
----- 22 6r 1A: r:	Transmit state watchdog time out in lin2_xmt_mgr. LIN channel number 2 thru 7
----- 22 6r 1B: r:	UART transmit buffer full or shift register not empty. LIN channel number 2 thru 7
----- 22 6r 1C:	LIN channel not enabled, xmt command not processed.

---

---

r:	LIN channel number 2 thru 7
-----	
22 6r 1D:	Transmit command, master/slave byte is neither.
r:	LIN channel number 2 thru 7
-----	
22 6r 1E:	not defined
r:	LIN channel number 2 thru 7
-----	
22 6r 1F:	not defined
r:	LIN channel number 2 thru 7
-----	
22 6r 20:	Stored pm length too long, in 7x 18 query.
r:	LIN channel number 2 thru 7
-----	
22 6r 21:	not defined
r:	LIN channel number 2 thru 7
-----	
22 6r 22:	not defined
r:	LIN channel number 2 thru 7
-----	
22 6r 23:	Invalid rcv buff0 state in lin2_new_byte.
r:	LIN channel number 2 thru 7
-----	
22 6r 24:	Invalid rcv buff1 state in lin2_new_byte.
r:	LIN channel number 2 thru 7
-----	
22 6r 25:	not defined
r:	LIN channel number 2 thru 7
-----	
...	
-----	
2x 70:	

---

---

...

-----  
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:

-----  
22 7F 0B:

-----  
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.

-----  
23 7F 21 yy: canB\_rcv\_mgr invalid object number CAN2.  
yy: object number

-----  
23 7F 22 yy: canB\_rcv\_mgr invalid object number CAN3.  
yy: object number

-----  
22 7F 23: canA\_rcv\_mgr invalid object number CAN0.

-----  
22 7F 24: canA\_rcv\_mgr invalid object number CAN1.

-----  
22 7F 25: canA\_rcv\_mgr invalid channel number.

-----  
22 7F 26: canA\_rcv\_mgr DLC too long.

-----  
22 7F 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.  
-----

23 7F 2D 0y: canB\_rcv\_mgr, a 'return(3)' from canB\_iso\_rcv\_mgr.  
'y' is CAN channel number.

-----

23 7F 2E yy: canB\_rcv\_mgr, CAN2, undefined frame number.  
yy: frame number.

-----

23 7F 2F yy: canB\_rcv\_mgr, CAN3, undefined frame number.  
yy: frame number.

-----

23 7F 30 yy: canB\_rcv\_mgr, CAN2, invalid transmit ack flag.  
yy: transmit ack flag.

-----

23 7F 31 yy: canB\_rcv\_mgr, CAN3, invalid transmit ack flag.  
yy: transmit ack flag.

-----

23 7F 32 0y: canB\_rcv\_mgr, invalid frame number.  
y: frame number.

-----

2x 80:

-----

2x 81:

-----

2x 82:

-----

2x 83:

-----

21 84: Command buffer mode fault.

-----

2x 85:

-----

23 86 xx yy:  
LIN1 error flags.  
b15:  
b14:

- b13:
- b12:
- b11:
- b10:
- b09:
- b08:
- b07: error in pit2 service code.
- b06: illegal receive buffer state.
- b05: break byte not 00.
- b04: received a byte, not a break, no buffer available.
- b03: received a byte, not a break, no active buffer.
- b02: synch byte not \$55.
- b01: received byte errors: RB, FE, PE, OE.
- b00: no receive buffer available.

-----  
2x 87:

-----  
2x 88:

-----  
2x 89:

-----  
2x 8A:

-----  
2x 8B:

-----  
2x 8C:

-----  
2x 8D:

-----  
2x 8E:

-----  
2x 8F:

-----  
2x 90:

-----

24 91 0z aa bb :

CANz error

z: CAN channel 2, 3.

aa bb: error flag bit map.

b15:

b14:

b13:

b12:

b11:

b10:

b9:

b8:

b7: bus off.

b6: bus warning.

b5: error counter ii overrun.

b4: FD protocol exception.

b3: RM protocol exception.

b2: transmit fifo overflow.

b1: receive fifo overflow.

b0: 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.

-----  
2x 94:

-----  
2x 95:

-----  
23 96 xx yy:

LIN0 error flags

b15:

b14:

b13:

b12:

b11:

- b10:
- b09:
- b08:
- b07: error in pit2 service code.
- b06: illegal receive buffer state.
- b05: break byte not 00.
- b04: received a byte, not a break, no buffer available.
- b03: received a byte, not a break, no active buffer.
- b02: synch byte not \$55.
- b01: received byte errors: RB, FE, PE, OE.
- b00: no receive buffer available.

-----

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)

-----

2x 99:

...

-----

22 E5 01: LIN0 transmit command too short.

-----

22 E6 01: KWP transmit command too short.

-----

22 E7 01: LIN1 transmit command too short.

-----

2x E8:

-----

2x E9:

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, KWP0, KWP1, and LIN2 thru LIN7**

63 01 0r 0z: Send received checksum to Client.  
r: channel: 7, 5, 8, 6, and A thru F.  
z: 0: disabled.  
1: enabled.

-----

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

63 02 0r zz: Receiver buffer timeout.  
r: channel: 7, 5, 8, 6, and A thru F.  
zz: time in milliseconds.

-----

**Not channel specific**

63 05 0r 0s: Digital output status.  
r: 0: output is high impedance.  
1: output is low impedance.  
s: 0: did NOT reset the time stamp counter.

---

1: did reset the time stamp counter.

-----  
**CAN2, CAN3**

63 06 0r 0s: Response format status.  
 r: channel: 2 or 3.  
 s: 0: response format depends on message length.  
 1: always use the long (12 xx yy) format.

-----  
**CAN0, CAN1, CAN2, CAN3, LIN0, LIN1, KWP0, KWP1, and LIN2 thru LIN7**

63 08 0r 0y: Time stamp status.  
 r channel: 0, 1, 2, 3, 7, 5, 8, 6, and A thru F.  
 y: 0 disabled.  
 1 enabled.

-----  
**LIN0, LIN1**

65 11 0y rr 0m tt: LIN mirror function set-up.  
 y: LIN channel, 5 or 7.  
 rr: receive ID to watch for.  
 m: 0 = transmit as slave.  
 1 = transmit as master.  
 tt: transmit ID.

-----  
**LIN0, LIN1**

63 12 0y 0s: LIN mirror function status command.  
 y: LIN channel, 5 or 7.  
 s: 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).

-----  
**KWP0, KWP1**

64 27 0r zz: P4 - transmit inter-byte time.)  
 r: channel: 8, 6.  
 zz: time in milliseconds.

-----  
**KWP0, KWP1**

64 28 0r 0z: Format Byte processing status.  
 r: channel: 8, 6.  
 z: 0: disabled.  
 1: enabled.

-----  
**KWP0, KWP1**

63 2A 0r ss: 'P3' – minimum inter-message time.  
 r: channel: 8, 6.  
 ss: time in milliseconds.

-----  
**Not channel specific**  
**(do not use the related command)**

63 31 xx yy: Return code from:  
 getsockopt(fd) (for query).  
 set or clear function called (for command).

-----  
**CAN0, CAN1, CAN2, CAN3, LIN0, LIN1, KWP0, KWP1, and LIN2 thru LIN7**

63 40 0r 0y: Send transmit acknowledgements to Client.  
 r channel: 0, 1, 2, 3, 7, 5, 8, 6, and A thru F.  
 y: 0 disabled.  
 1 enabled.  
 2 echo enable.  
 (option 2 only valid for channels CAN2 and CAN3.)

-----  
**KWP0, KWP1**

64 46 0r ss tt: Fast Init 'W5' K-line idle time.  
 r: channel: 8, 6.  
 ss tt: time in milliseconds.

-----  
**KWP0, KWP1**

63 47 0r ss: Fast Init K-line low time.  
 r: channel: 8, 6.  
 ss: time in milliseconds.

-----  
**KWP0, KWP1**

63 48 0r ss: Fast Init K-line high time.  
 r: channel: 8, 6.  
 ss: time in milliseconds.

-----  
**KWP0, KWP1**

63 4B 0r 0z: KWP checksum method.  
 r: channel: 8, 6.  
 z: 0: no transmit checksum.  
 1: sum of bytes.  
 2: sum of bytes, 2's complement.  
 3: XOR of bytes.

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

63 50 0r 0z: Baud rate.  
 r: channel: 7, 5, 8 6, and A thru F.  
 z: 1: 2400 baud.  
 2: 9600 baud.  
 3: 19200 baud.  
 4: 10400 baud.

64 50 0r yy zz: Baud rate.  
 r: channel: 7, 5, 8 6. (Not available for channels A thru F.)  
 yy zz: divisor load.  
 $125,000,000 / 32 / yy\ zz$   
 (where 'yy zz' is converted to decimal)

-----  
**LIN0, LIN1, and LIN2 thru LIN7**

63 52 0r zz: Maximum frame time.  
 r: channel: 7, 5, and A thru F.  
 zz: time in milliseconds.

-----  
**LIN0, LIN1, and LIN2 thru LIN7**

63 5A 0r 0z: Checksum type.  
 r: channel: 7, 5, and A thru F.  
 z: 0: classic (LIN 1.3).  
 1: enhanced (LIN 2.0).  
 2: none.

-----  
**LIN0, LIN1, and LIN2 thru LIN7**

63 66 0r 0z: "ID byte only" error message to Client.  
 r: channel: 7, 5, and A thru F.  
 z: 0: disabled.  
 1: enabled.

**KWP0, KWP1**

63 66 0r 0z: Send "One byte only" error message to Client.  
 r: channel numbers: 8, 6.  
 z: 0: disabled.



---

b14:	'1'	reserved, not used.	
b13:	'1'	reserved, not used.	
b12:	'1'	reserved, not used.	
b11:	'1'	reserved, not used.	
b10:	'1'	reserved, not used.	
b09:	'1'	reserved, not used.	
b08:	'1'	reserved, not used.	
b07:	CAN3	transceiver status	0 = standby / disabled. 1 = operate / enabled.
b06:	CAN2	transceiver status	0 = standby / disabled. 1 = operate / enabled.
b05:	CAN1	transceiver status	0 = standby / disabled. 1 = operate / enabled.
b04:	CAN0	transceiver status	0 = standby / disabled. 1 = operate / enabled.
b03:	CAN3	termination	0 = termination disabled. 1 = termination enabled.
b02:	CAN2	termination	0 = termination disabled. 1 = termination enabled.
b01:	CAN1	termination	0 = termination disabled. 1 = termination enabled.
b00:	CAN0	termination	0 = termination disabled. 1 = termination enabled.

7: Initialization attempt responses.

-----  
**KWP0, KWP1**

73 13 0r ss: 'Fast Init' (ISO 14230) responses  
 r: channel number: 8, 6.  
 ss: 00 – init attempt failed.  
 01 – init sequence started.  
 11 – init sequence completed successfully.

8: CAN configuration reports.

-----  
**CAN2, CAN3**

**(do not use this command)**

84 01 0r yy zz: Status of channel ISO15765 buffer watchdog time-out value.  
 r: channel: 2, 3.  
 yy zz: reset time in msec.

-----  
**CAN2, CAN3**

**(only use this command when you are certain you need to)**

83 02 0r 0z: Status of type of CAN-FD operations.

r: channel: 2, 3.  
 z: 0 = ISO operations.  
 1 = Bosch operations.

-----  
**CAN0, CAN1**

84 04 0r 0z 0y: Configuration of CAN object.  
 r channel: 0, 1.  
 z: object number: \$0 to \$F.  
 y: 0: object disabled.  
 1: object enabled for receive.  
 2: object enabled for transmit. (CAN0/1 only.)

-----  
**CAN2, CAN3**

84 04 0r zz 0y: Configuration of CAN object.  
 r channel: 2, 3.  
 zz: object number: 0x00 to 0x3F.  
 y: 0: object disabled.  
 1: object enabled for receive.  
 2: object enabled for transmit. (CAN0/1 only.)

-----  
**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.

-----  
**CAN2, CAN3**

**(do not use this command)**  
 82 09 yy: Maximum count of received frames processed  
 before exiting the CAN receive manager.  
 yy: count.

-----  
**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.

-----  
**CAN0, CAN1, CAN2, CAN3**

**(only valid if hardware has been modified)**

83 13 0r 0s: Set CAN transceiver status.  
 r: channel: 0, 1, 2, 3.  
 s: 0 = set 2-wire transceiver to standby (disabled).  
 1 = set 2-wire transceiver to operate (enabled).

-----  
**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.

86 17 0r w0 zz 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)  
 zz: object number: 0x00 to 0x3F.

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.

88 17 0r w0 zz 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)  
 zz: object number: 0x00 to 0x3F.  
 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.

**CAN2, CAN3**

11 bb 3r y0 pp tt vv ww zz mm nn ...:

11: long header.  
 bb: count of bytes to follow.  
 3: long periodic message response.  
 r: channel number: 2, 3.  
 y: b7: IDE 1: 29-bit.  
     0: 11-bit.  
 b6: RTR 1: is an RTR frame.  
     0: is NOT an RTR frame.  
 b5: EDL 1: CAN-FD.  
     0: Classical CAN.  
 b5: BRS 1: high speed data field.  
     0: normal speed for entire frame.  
 0  
 pp: message number: \$00 to \$2F.  
 tt vv: 11-bit id.  
 tt vv ww zz: 29-bit id.  
 mm nn ... : data field (0 to 64 bytes, inclusive).

**LIN0, LIN1, and LIN2 thru LIN7**

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

**KWP0, KWP1**

8x 18 0r 0p ss tt vv...: Periodic message set-up.

r: channel: 8, 6.  
 p: message number: \$0 to \$F.  
 ss tt vv... : data field.

**CAN0, CAN1**

84 19 0r pp 0y: Object assigned to CAN periodic message.

r: Channel: 0, 1.  
 pp: Message number: \$00 to \$2F.  
 y: Object number: \$0 to \$F.

**CAN0, CAN1, CAN2, CAN3**

84 1A 0r pp 0v: Periodic message disable/enable status.

r: channel: 0, 1, 2, 3.  
 pp: message number: \$00 to \$2F.  
 v: 0 disabled.  
 1 enabled.

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

84 1A 0r 0p 0v: Periodic message disable/enable.

r: channel: 7, 5, 8, 6, and A thru F.  
 p: message number: \$0 to \$F.  
 v: 0 disabled.  
 1 enabled.

**CAN0, CAN1, CAN2, CAN3**

85 1B 0r pp vv ww: Periodic message interval count.

r: channel: 0, 1, 2, 3.  
 pp: message number: \$00 to \$2F.  
 ww vv: interval count.

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

85 1B 0r 0p vv ww: Periodic message interval count.

r: channel: 7, 5, 8, 6, and A thru F.  
 p: message number: \$0 to \$F.  
 vv ww: time in milliseconds.

**CAN0, CAN1, CAN2, CAN3, LIN0, LIN1, KWP0, KWP1, and LIN2 thru LIN7**

82 1C 0r: All periodic message for channel 'r' disabled.

r: channel: 0, 1, 2, 3, 7, 5, 8, 6, and A thru F.  
 r: 'FF' – all messages, all channels disabled.

This command was changed with version '0049'.

-----  
**CAN2, CAN3**

83 1F 0r 0s: Status of processing of received CAN frames.  
 r: channel: 2, 3.  
 s: 0 = do not process received CAN frames (discard them).  
 1 = process received CAN frames.

-----  
**CAN0, CAN1, CAN2, CAN3**

85 20 0r 00 0s tt: 'Mirror' function receive 11-bit ID.  
 r: channel: 0, 1, 2, 3.  
 00.  
 stt: 11-bit ID.

87 20 0r 00 ss tt vv ww: 'Mirror' function receive 29-bit ID.  
 r: channel: 0, 1, 2, 3.  
 00.  
 ss tt vv ww: 29-bit ID.

-----  
**CAN0, CAN1, CAN2, CAN3**

85 21 0r 0p 0s tt: 'Mirror' function transmit 11-bit ID.  
 r: channel: 0, 1, 2, 3.  
 p: object number: \$0 to \$F.  
 stt: 11-bit ID.

87 21 0r 0p ss tt vv ww: 'Mirror' function transmit 29-bit ID.  
 r: channel: 0, 1, 2, 3.  
 p: object number: \$0 to \$F.  
 ss tt vv ww: 29-bit ID.

-----  
**CAN0, CAN1, CAN2, CAN3**

83 22 0r 0s: 'Mirror' function status.  
 r: channel: 0, 1, 2, 3.  
 s: 0 = disable.  
 1 = enable.

-----  
**CAN0, CAN1, CAN2, CAN3**

83 25 0r ss: The flow control additional separation time.  
 r: channel: 0, 1, 2, 3.  
 ss: additional time, in milliseconds,  
 to add to the responding node separation time.

**CAN0, CAN1**

83 27 0r 0z 00: ISO 15765 padding is disabled.  
 r: channel: 0, 1.  
 z: object number: \$0 to \$F.  
 00: disabled.

84 27 0r 0z 01 ww: ISO 15765 padding is enabled.  
 r: channel: 0, 1.  
 z: object number: \$0 to \$F.  
 01: enabled.  
 ww: pad byte.

**CAN2, CAN3**

83 27 0r zz 00: ISO 15765 padding is disabled.  
 r: channel: 2, 3.  
 zz: object number: 0x00 to 0x3F.  
 00: disabled.

84 27 0r zz 01 ww: ISO 15765 padding is enabled.  
 r: channel: 2, 3.  
 zz: object number: 0x00 to 0x3F.  
 01: enabled.  
 ww: pad byte.

**CAN0, CAN1**

84 28 0r 0y 0s: Objects are paired; ISO 15765 operations are enabled. 'AE' disabled.  
 r: channel: 0, 1.  
 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.  
 y: object number: \$0 to \$F.  
 s: object number: \$0 to \$F.  
 ww: 'AE' byte.

**CAN2, CAN3**

84 28 0r yy ss: Objects are paired; ISO 15765 operations are enabled. 'AE' disabled.  
 r: channel: 2, 3.  
 yy: object number: 0x00 to 0x3F.  
 ss: object number: 0x00 to 0x3F.

85 28 0r yy ss ww: Objects are paired; ISO 15765 operations are enabled. 'AE' enabled.  
 r: channel: 0, 1, 2, 3.  
 yy: object number: 0x00 to 0x3F.  
 ss: object number: 0x00 to 0x3F.

ww: 'AE' byte.

-----  
**CAN2, CAN3**

84 29 0r yy ss: 'max\_dlc'. (Only used with ISO15765 processing.)  
 r: channel: 2, 3.  
 yy: object number: 0x00 to 0x3F.  
 ss: only the following values are valid (hex digits):  
 08, 0C, 10, 14, 18, 20, 30, 40.

-----  
**CAN0, CAN1**

85 2A 0r yz ss tt: Object 11-bit ID.  
 r: channel: 0, 1.  
 y: b7: 0.  
 b6: RTR.  
 0: normal frame.  
 1: RTR true, remote transmit request.  
 b5: 0  
 b4: 0  
 z: object number: \$0 to \$F.  
 ss tt: 11-bit ID.

87 2A 0r yz ss tt vv ww: Object 29-bit ID.  
 r: channel: 0, 1.  
 y: b7: 0.  
 b6: RTR.  
 0: normal frame.  
 1: RTR true, remote transmit request.  
 b5: 0  
 b4: 0  
 z: object number: \$0 to \$F.  
 ss tt vv ww: 29-bit ID.

-----  
**CAN2, CAN3**

85 2A 0r yz ss tt: Object 11-bit ID.  
 r: channel: 2, 3.  
 y: b7: 0.  
 b6: RTR.  
 0: normal frame.  
 1: RTR true, remote transmit request.  
 b5: 0  
 b4: 0  
 z: object number: \$0 to \$F.  
 ss tt: 11-bit ID.

86 2A 0r y0 zz ss tt: Object 11-bit ID.  
 r: channel: 2, 3.  
 y: b7: 0.  
 b6: RTR.  
 0: normal frame.  
 1: RTR true, remote transmit request.  
 b5: 0  
 b4: 0  
 zz: object number: 0x00 to 0x3F.  
 ss tt: 11-bit ID.

87 2A 0r yz ss tt vv ww: Object 29-bit ID.  
 r: channel: 2, 3.  
 y: b7: 0.  
 b6: RTR.  
 0: normal frame.  
 1: RTR true, remote transmit request.  
 b5: 0  
 b4: 0  
 z: object number: \$0 to \$F.  
 ss tt vv ww: 29-bit ID.

88 2A 0r y0 zz ss tt vv ww: Object 29-bit ID.  
 r: channel: 2, 3.  
 y: b7: 0.  
 b6: RTR.  
 0: normal frame.  
 1: RTR true, remote transmit request.  
 b5: 0  
 b4: 0  
 zz: object number: 0x00 to 0x3F.  
 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.

86 2C 0r y0 zz ss tt: 11-bit mask.  
 r: channel: 2, 3.  
 y: b7: IDE bit.  
     b6: 0.  
     b5: EDL bit.  
     b4: 0  
 zz: object number: 0x00 to 0x3F.  
 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.

88 2C 0r y0 zz ss tt vv ww: 29-bit mask.  
 r: channel: 2, 3.  
 y: b7: IDE bit.  
     b6: 0.  
     b5: EDL bit.  
     b4: 0  
 zz: object number: 0x00 to 0x3F.  
 ss tt vv ww: mask value, 29-bit.  
     1: bit must match.

---

0: bit is don't care.

-----  
**CAN0, CAN1**

84 30 0r 0z 0s: Disable / Enable ISO 15765 'AE' operation.  
 r: CAN channel 0, 1.  
 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.  
 z: object number: \$0 to \$F.  
 s: 0 = disabled, 1 = enabled.  
 ww: 'ae' byte.

**CAN2, CAN3**

84 30 0r zz 0s: Disable / Enable ISO 15765 'AE' operation.  
 r: CAN channel 2, 3.  
 zz: object number: 0x00 to 0x3F.  
 s: 0 = disabled, 1 = enabled.

85 30 0r zz 0s ww: Disable / Enable ISO 15765 'AE' operation; specify 'AE' byte.  
 r: CAN channel 2, 3.  
 zz: object number: 0x00 to 0x3F.  
 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**

85 50 0y 0z 0r ss: Frame buffer 11-bit arb ID.  
 y: channel: 2, 3.  
 z: frame buffer number: 0x00 to 0x0F.  
 r ss: 11-bit arb ID.

87 50 0y 0z rr ss tt vv: Frame buffer 29-bit arb ID.  
 y: channel: 2, 3.  
 z: frame buffer number: 0x00 to 0x0F.  
 rr ss tt vv: 29-bit arb ID.

-----  
**CAN2, CAN3**

85 51 0y 0z rr ss: Frame buffer timer reset value.

---

y: channel: 2, 3.  
 z: frame buffer number: 0x00 to 0x0F.  
 rr ss: timer reset value in msec.

-----  
**CAN2, CAN3**

84 52 0y 0z 0r: Frame buffer mode.  
 y: channel: 2, 3.  
 z: frame buffer number: 0x00 to 0x0F.  
 r: 0 = FIFO mode.  
 1 = Rolling mode.

-----  
**CAN2, CAN3**

85 53 0y 0z rr ss: Frame buffer slot status  
 y: channel: 2, 3.  
 z: frame buffer number: 0x00 to 0x0F.  
 rr: number of full slots.  
 ss: number of empty slots.

-----  
**CAN2, CAN3**

84 54 0y 0z 0r: Frame buffer status.  
 y: channel: 2, 3.  
 z: frame buffer number: 0x00 to 0x0F.  
 r: 0 = disabled.  
 1 = enabled.

-----  
**CAN2, CAN3**

84 5F 0r ss 0z: RC6 status.  
 r: channel: 2, 3.  
 ss: periodic message number (0x00 to 0x1F).  
 y: 0 = disabled.  
 1 = enabled, use KA\_01.  
 2 = enabled, use Klemmen\_Status\_01.  
 3 = enabled, use Motor\_Code\_01.  
 4 = enabled, use UserStateServer\_01.  
 5 = enabled, use 'user defined'. (not implemented, yet)

-----  
**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.

9x 01 00:

...

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.

9x 01 22:

...

- 
- 92 01 30: AVT-424 hardware reset complete.  
(This report is generated by the AVT-423.)
  - 92 01 31: AVT-424 firmware reset complete.
  - 93 04 xx yy: AVT-423 firmware version report. Version is xx yy.
  - 93 05 xx yy: AVT-424 firmware version report. Version is xx yy.
  - 93 06 xx yy: Mezzanine board model number. AVT-424: xx yy = 04 24.
  - 91 0A: Power-on and 'F1 C3' reset.
  - 91 0F: Response to an 'F1 A5' reset.
  - 93 28 0x yz: Model number report. xyz is the model number.
  - 91 3A: AVT-423 Ethernet connect response.
  - 94 3B xx yy zz: CAN-FD IP core version number report.
  - 97 3C pp rr ss tt vv ww: AVT-423 MAC address.

A: \_\_\_\_\_

B: \_\_\_\_\_

C: \_\_\_\_\_

D: \_\_\_\_\_

E: \_\_\_\_\_

F: \_\_\_\_\_

---

## 18. Appendix A - AVT-424 LIN Expansion Board

The AVT-424 LIN Expansion board is a (relatively) small board designed to be mounted to the top of an AVT-423 board. It is a mezzanine or daughter board.

### Summary

- It adds six (6) fully functional LIN channels to the capabilities of the AVT-423 board. They are designated LIN2 thru LIN7 (channels 'A' thru 'F').
- Each channel is capable of: receive, transmit, periodic messages, and slave periodic messages.
- Each channel is completely independent of all other channels (LIN, CAN, ...).
- Each channel requires a separate external supply for the transceiver.
- The AVT-424 uses a DA-15P connector for all external LIN bus connections and supplies.
- The Client communicates with the AVT-424 through the AVT-423. A separate Ethernet connection is not required.
- The Client controls all the new LIN channels using the same commands as channels LIN0 and LIN1. The Commands and Responses (Sections 17 and 17.1) have been updated to indicate the command changes for the additional LIN channels.

### Limitation

The AVT-423 firmware does not know if an AVT-424 board is present or not.

If the Client sends a command to the AVT-424 and an AVT-424 board is not present – no response will be received.

### Overview

The AVT-424 uses a PIC24FJ microcontroller to implement six (6) channels of LIN communications.

Each LIN channel is:

- Completely independent.
- Supports LIN revisions: 1.2, 1.3, 2.0, 2.1, 2.2A.
- Provides 16 periodic messages including slave periodic messages.
- Uses all the same commands as channels LIN0 and LIN1 of the AVT-423.

Each LIN bus:

- Uses an NXP (Freescale) MC33660 transceiver.
- Has a separate and independent external VBATT supply.
- The operating voltage range for each VBATT supply is: +8 to +18 VDC.

- Bus idle VBATT current consumption is ~ 1 milliamp.
- Has a 1 K ohm pull-up resistor to the VBATT supply.
- Is reverse polarity protected.

Internal Connections

The AVT-423 and AVT-424 share a common ground.

All ground (or return) pins on the DB-25P and DA-15P connectors are electrically connected together on the two boards.

The AVT-423 board supplies +3.3 VDC and +5.0 VDC power to the AVT-424 board.

Processors on the AVT-423 and AVT-424 communicate using a local high speed serial link. All communications and handshaking signals are routed through board-to-board connectors:

- P10 on the AVT-423 board.
- P3 on the AVT-424 board.

External Connections

Connector P10 on the AVT-424 board is a DA-15P connector to “connect to the world”.

The pin-out for P10 is listed in Table 3, below.

<u>Pin #</u>	<u>Description</u>	<u>Notes</u>
1	LIN2 bus	channel \$A
9	LIN2 bus supply	VBATT supply and reference for LIN2 transceiver only
2	LIN3 bus	channel \$B
10	LIN3 bus supply	VBATT supply and reference for LIN3 transceiver only
3	LIN4 bus	channel \$C
11	LIN4 bus supply	VBATT supply and reference for LIN4 transceiver only
4	LIN5 bus	channel \$D
12	LIN5 bus supply	VBATT supply and reference for LIN5 transceiver only
5	LIN6 bus	channel \$E
13	LIN6 bus supply	VBATT supply and reference for LIN6 transceiver only
6	LIN7 bus	channel \$F

14	LIN7 bus supply	VBATT supply and reference for LIN7 transceiver only
7	do not use	
15	ground	same as AVT-423
8	do not use	

P10 (the DA-15P connector on the AVT-424 board)

Table 3

Connector P1 on the AVT-424 is a 6 pin vertical pin header.  
This connector is used by the factory for testing.  
DO NOT connect anything to P1.

## **19. 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).  
fIndex 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.
- 0030 (A) Section about multiple simultaneous Client connections.
- Section about saving startup parameters in non-volatile memory.
- Added '5x 80' command for storing and reading start-up parameters stored in non-volatile memory.
- 0030 (B) Corrected two errors in Section 11.11.3.1, the baud rate command and the mask command.
- Corrected two errors in Section 11.11.4.1, the baud rate command and the mask command.
- 0032 (A) Added notes about the new 'mirror' function. Added all 'mirror' commands and responses as well as two new error messages.

- 
- Changed all “FPGA” references to “CAN-FD IP core”.  
 Added the ‘B1 03’ query for model number.  
 Added the ‘B1 04’ query for MAC address.  
 Noted that the ‘F0’ command is deprecated and will be removed in the future.
- 0033 (A) Added notes, commands, and responses about LIN mirror function. Corrected several ‘22 7F xx’ error messages.
- 0035 (A) Changed the response to an ‘F1 A5’ reset command from ‘91 3A 03 04 xx yy’ to ‘91 0F’.  
 Added section 9.3 about the different automatic responses sent to the Client.  
 Added section 9.4 briefly explaining the differences between the two software reset commands.
- 0036 (A) Updated the responses the Client receives for power on, software reset, and Client connection. This was done to remove ambiguity as to the source of the various responses.
- 0040 (A) For firmware versions 0037, 0038, and 0039 - most changes were to firmware to improve performance and reliability of channels CAN2 and CAN3. The only changes to the manual were in a number of ‘2x 5F yy’ error responses.
- 0040 (B) Corrections to ‘77 2A’ and ‘77 2C’ command examples on page 29.
- 0043 (A) Lots of updates.  
 For channels CAN2 and CAN3 added the ability to use 64 (decimal) objects, numbered 0x00 to 0x3F.  
 Updated numerous commands to support that change.  
 Added some text explaining it.  
 Added KWP support for both physical K-lines. Now each physical K-line can support either LIN or KWP.  
 Added some commands to support KWP mode.
- 0043 (B) Added the CAN2/3 transmit command formats to support all of the objects.
- 0049 (A) Added information about KWP periodic message support and Fast Init support.  
 Added Table 2 with connection information for rev. “F” board.  
 Added notes about JP1 and JP2 – separate supplies for the two LIN transceivers.  
 Added information about the new AVT-424 LIN Expansion board – Appendix A.  
 Changed all references to “client” to “Client”.  
 Strikethrough “Flexray” references in the “Commands” and “Responses” sections.  
 I plan to remove all “Flexray” references in a future revision of this manual.
- 0053 (A) Added the ‘7x 25’ command introduced with version ‘0050’.  
 Added several new error codes introduced with version ‘0053’.
- 0054 (A) Added support for ‘long’ periodic messages for channels CAN2 and CAN3.  
 Added the ‘11 bb’ or ‘12 00 bb’ command formats to both define and query for long periodic messages.
- 0058 (A) Added information about changes in firmware versions ‘0055’, ‘0056’, ‘0057’, and ‘0058’.
-

0055: Corrections to the '7x 17' command.

0056: Reduced the number of CAN periodic messages from 48 to 32 (decimal).  
They are numbered: 0x00 to 0x1F.  
Implemented CAN2 and CAN3 'frame buffers' along with all related commands.

0057: Implemented CAN2 and CAN3 transmit echo feature.  
Lots of 'internal' changes to timer update routines.  
Lots of 'internal' changes to CAN2 and CAN3 interrupt routines, frame numbers, and related issues.  
The '7x 08' command is deprecated and will not be updated.  
I plan to remove the '7x 08' command at some point in the future.

0058 (B) Minor text corrections to the '7x 2A' command.

0068 (A) Corrections, added several commands, added a CAN periodic message special function.  
Removed: '7x 05' command.  
Removed: '7x 08' command.

## 20. 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/download.htm#AVT-423>

Firmware Version Descriptions

[http://www.AVT-HQ.com/423\\_sw.htm](http://www.AVT-HQ.com/423_sw.htm)