4 December 2005

A Discussion about CAN

The AVT-418, AVT-512, AVT-717, and AVT-718 all support a single channel of CAN (Controller Area Network) operations. This document provides a brief description of how the CAN network interface on our interface units is implemented and used.

Included in this document are details on some of the CAN commands and technical implementation details with respect to the CAN network and AVT interface units.

Reference Documentation.

The reader should consult other sources for detailed information on the architecture, operation, and implementation of CAN networks.

Available from Intel is the PDF document "82527 Serial Communications Controller Architectural Overview." (File name: 27241003.PDF) This document contains very good information on the operation of a CAN network, details on the global masks, and setting up and using the objects.

Test and Development Considerations.

CAN operations require a network consisting of a minimum of two nodes in order to transmit a message. This becomes important in development, integration, and testing.

Silicon

All AVT interface products use the Intel 82527 CAN controller device to implement the CAN network interface.

Physical Layer

The AVT-418 and AVT-718 supports three different physical layer implementations of CAN.
- 2-wire CAN (also known as: ISO 11898, J1939, or CAN-C).
- 2-wire CAN (also known as: ISO 11519 or CAN-B).
- Single Wire CAN (known as: J2411 or SWC).

The AVT-512 and 717 support two implementations for the physical layer of CAN.
- 2-wire CAN (also known as: ISO 11898, J1939, or CAN-C)
- Single Wire CAN (known as: SWC and J2411.)

The original Bosch CAN specification does not specify the baud rate or physical layer.

Physical Layer

The most common implementation is a balanced signal or 2-wire network (known as ISO 11898, J1939, CAN-C and possibly other names). For this implementation the Philips 82C250 transceiver is used. Later versions of AVT-718 and AVT-418 hardware use the Philips TJA1050T transceiver for better performance and EMI considerations. Some AVT-512 units may be equipped with Philips 82C251 transceiver.

The physical layer to be used is selected through a software command.
Refer to the "Master Commands and Response" document for the command specifics.
Refer to the individual unit User Manual for signal and pin definitions.

What are the different parts of a CAN message ?

A CAN message consists of a maximum of 108 bits (CAN 1.0) or 128 bits (CAN 2.0B). All of the bits are identified in the specification. Only the "Arbitration" and "Data" fields are of interest.

[A diagram of a CAN 2.0B message frame is included at the end of this document. The diagrammed frame references SAE specification J1939 and uses the term PDU in place of arbitration field.]

Arbitration Field.

In the automotive industry, CAN arbitration fields are more commonly known as "Message IDs".

The arbitration field serves to define the priority of a message relative to all other messages. In CAN a 'zero' bit has higher priority than a 'one' bit. Therefore, a message with a lower value arbitration field has a higher priority than a message with a higher value arbitration field.

The arbitration field can be used to indicate what kind of data the message contains. It can also be used to indicate the "to" address for the message.

Data Field.

The data field of a message can contain from zero to eight bytes of user defined data.

What kinds of CAN are there ?

There are two major versions of CAN. They are CAN 1.0 and CAN 2.0B.

- CAN 1.0 messages have 11-bit arbitration fields.
- CAN 2.0B messages have 29-bit arbitration fields.
- Both versions can coexist on the same network.
- AVT CAN interface units can support operations for either message type.
- A node can only transmit and/or receive messages for which it is set up.

On AVT interface boards, there are fifteen "objects." Each object is independent, can support either format, and can be setup as required.

What is an Object ?

The AVT interface unit (using the Intel device) communicate on the CAN network using what are known as objects. Each object can be thought of as a ‘mailbox’ through which a message is transmitted or received. Each object, or mailbox, can only transmit or receive; not both simultaneously.

The Intel device has 15 objects; numbered 1 to 15 ($1 to $F) available.
Object 15 ($F) can only conduct receive operations.

Objects 1 to 14 can be configured for either transmit or receive operations (one object can not support simultaneous receive and transmit functions). Each object must be configured properly and enabled before it can be used.

What is Configured Automatically ?

The AVT interface automatically configures some CAN parameters when CAN mode is entered or when the CAN device is commanded to be reset ($21 $06). Some of the parameters are listed here.

- The input clock is: 16 MHz.
- The DSC bit is set to 1.
  (This sets the SCLK [system clock] to 8 MHz.)
- Both Global Masks are set to all 1’s.
  (This is a ‘must match’ condition.)
- All objects are disabled and ID’s are set to all 1’s.
- The CAN network data rate (or speed) is set to 250 kbps.
  (Details on network speed setting are provided later in this document.)
- On the AVT-418 and AVT-718 the interface is physically disconnected from the network.

What do I have to Configure ?

The user must initialize the following parameters to utilize the CAN network.

- Global Mask - 11-bit (only if necessary).
- Global Mask - 29-bit (only if necessary).
- Each object that is to be used.
- The network baud rate.
- Select the physical layer.

How is a message received ?

When a message is received it’s ID is matched against the arbitration fields of all objects that are enabled, setup for receiving, and have the same arbitration field length. If the message ID matches an object arbitration field, that message is received by that object and forwarded to the host computer.

What is a Global Mask ?

The ID of a received message (including a message being transmitted) is passed through a global message mask (logical AND) before being compared to the ID of each enabled object.

There are two global message masks. One is for 11-bit arbitration field messages. The other is for 29-bit arbitration field messages.

The bits in the global masks determine what bits in a message ID are “must match” or “don’t care” when compared to an active object’s arbitration field. A ‘0’ bit in the global mask designates that bit as a “don’t care” bit when compared to an object arbitration field. A ‘1’ bit in the global mask means that bit in the message ID must match the object arbitration field. A global mask set to all 1’s means the incoming message ID must match exactly the object arbitration field.

The proper use of the global message masks permits the user to set up an object to receive a ‘class’ or ‘group’ of messages through a single object.

What commands are available ?

Refer to the “Master Commands and Responses” document, CAN operations section. Document file name is MASTERxx.PDF (where xx is the version number of the document) and is available from the App Notes section of the Download page of our web site. [http://www.AVT-HQ.com/download.htm]

What do I configure in an object ?

The user must set several parameters of the object.
(Examples are provided for both transmit and receive operations.)

- Arbitration field or Message ID.
- Transmit or receive.
- [Note that data field length is not important, is actually ignored, should be set to 00.]
- Enable or disable the object.

Arbitration fields are not even bytes in length. How do I use them ?

The arbitration fields are either 11-bits or 29-bits in length. When setting either, the upper most bits are zeros. Examples:
An 11-bit arbitration field is: 101 0110 1011. In hex digits: 05 6B.
A 29-bit arbitration field is: 1 1001 1011 0111 1000 0101 1101 0110. In hex digits: 19 B7 85 D6

Network Speed

The AVT-418, 512, 717, and 718 support two methods to set the network baud rate:
Preprogrammed and user defined.

There are numerous preprogrammed baud rates that can be chosen using the 72 0A xx command.

The table below contains the available baud rates as well as the values that are written to the two bit timing registers. This is provided as a reference to the user. Register values are in hex.

72 0A xx Command	Baud Rate	BT0	BT1
00	user defined		
01	1 Mbps	00	23
02	500 kbps	C0	D8
03	250 kbps	C1	D8
04	125 kbps	C3	D8
05	100 kbps	C4	D8
06	50 kbps	C9	D8
07	41.67 kbps	CB	D8
08	25 kbps	4F	CD
09	80 kbps	44	CD
0A	33.333 kbps	CB	AF
0B	83.333 kbps	87	98

The user can also write directly to the two bit timing registers to set a desired baud rate and characteristics. This is done using the 73 0B xx yy command. When this method is used the baud rate query will return: 82 0A 00. The ‘00’ value indicates that the user has set the baud rate by writing directly to Bit Timing Registers 0 and 1.

Ford HS-CAN [High Speed CAN] 500 kbaud. The following baud rate command is recommended: 73 0B 40 2B (instead of the 72 0A 02 setting listed above).

Ford MS-CAN [Medium Speed CAN] 250 kbaud. The following baud rate command is recommended: 73 0B C1 58 (instead of the 72 0A 03 setting listed above).

Ford MS-CAN [Medium Speed CAN] 125 kbaud. The following baud rate command is recommended: 73 0B C3 58 (instead of the 72 0A 04 setting listed above).

Baud Rate Technical Details

A CAN network can be operated at nearly any baud rate (or speed) up to 1 Mbps. The actual bit time on a CAN network (using the 82527 device) is determined primarily by three parameters: Baud rate prescaler, TSEG1, and TSEG2. The operating bit time (baud rate) is determined by the values these parameters are programmed to. For detailed information consult the “Architectural” document referenced previously.

Bit timing registers 0 and 1 contain the three primary parameters that control the bit time (or inversely, the baud rate). These are listed here with the allowable ranges of values.

Baud rate prescaler	TSEG1	TSEG2
0 to 63	2 to 15	1 to 7

A Demonstration

Any two AVT CAN interface units can be set up, connected together, and used to become familiar with CAN network and AVT interface operations.

The following listing provides a step by step example of how to set up two units, identically, so that messages can be sent from one to the other.

Interface Unit Connections

- Connect the CAN_H and CAN_L signals of both interface units together.
- Supply power to both interface units.
- A separate host computer may be used for each interface unit. Otherwise, connect the host computer to one interface unit, set it up, and then switch the host computer to the other interface unit.

Interface Unit Configuration

Configure both interface units as described here.
All commands are hex digits.
All commands are shown. Responses are in *italics*.
Explanations for each command are provided.

Always select the baud rate before selecting the physical layer. This will usually prevent error messages from being generated and possibly prevent the generation of error conditions on the network.

⇒ F1 A5 --> *91 12, 92 04 xx*
Reset the interface unit.

⇒ E1 99 --> *91 10, 82 11 00*
Enter CAN mode, CAN mode response, no physical layer selected.

⇒ 72 0A 03 --> *82 0A 03*
250 kbps baud rate.

⇒ 72 11 02 --> *82 11 02*
Two wire CAN mode.

*** Set up Object #1 as a transmit object. ***

⇒ 77 05 01 10 01 05 03 57 --> *87 05 01 10 01 00 03 57*
Object setup, object #1, transmit, 11-bit arbitration field, data length = 00, ID = 03 57.

⇒ 77 06 01 F1 E2 D3 C4 B5 --> *87 06 01 F1 E2 D3 C4 B5*
Object data setup, object #1, data = F1 E2 D3 C4 B5.
Setting the object data field updates the data length.

⇒ 73 04 01 10 --> *83 04 01 01*
Object status, object #1, enabled as transmit.

*** Set up object #2 as a receive object. ***

⇒ 77 05 02 01 01 00 03 57 --> *87 05 02 01 01 00 03 57*
Object setup, object #2, receive, 11-bit arbitration field, data length = 00, ID = 03 57.

⇒ 73 04 02 01 --> *83 04 02 01*
Object status, object #2, enabled as receive.

*** Transmit a message from one interface unit. ***

⇒ 73 07 01 01
Transmit object #1 from one interface unit.
The other interface unit will receive the message in object #2.

Two responses from the transmitting interface unit:
--> *83 07 01 01*
Confirmation of the transmit command.
--> *82 09 01*
Object #1 successfully transmitted.

*** Receive a message at the other interface unit. ***

⇒ Receive the response:
--> *08 02 03 57 F1 E2 D3 C4 B5*
Received message from the network, through object #2, ID = 03 57, data bytes follow and are F1 E2 D3 C4 B5.

A Transmit Example

The following example demonstrates setting up an object for transmission and then transmitting a message. Object #6 is used here.

This example is not related to the previous demonstration.

All commands are shown. Responses are in *italics*.

⇒ Set up the Global Message Masks, as required. (Default status is recommended.)

11-bit mask:	73 01 xx yy --> *83 01 xx yy*
29-bit mask:	75 02 rr ss tt vv --> *85 02 rr ss tt vv*

⇒ Disable the object (if necessary).
73 04 06 00 --> *83 04 06 00*

⇒ Set the object configuration.
77 05 06 10 01 00 03 C4 --> *87 05 06 10 01 03 03 C4*
Explanation:

77 05:	Command (upper nibble 7), byte count (lower nibble 7), and command type (05).
06:	Object number.
10:	Transmit.
01:	Standard arbitration field (11 bits).
00:	Data length (not used).
03 C4:	Arbitration field.

⇒ Enable the object.
73 04 06 10 --> *83 04 06 10*

⇒ Set the object data to be transmitted.
75 06 06 1A 2B 3C --> *85 06 06 1A 2B 3C*
Explanation:

75 06:	Header byte, command, and command type.
06:	Object number.
1A 2B 3C:	Actual message data.

⇒ Transmit the message.
73 07 06 01 --> *83 07 06 01*
This response indicates that the command was executed only.

⇒ Notification that the message has been successfully transmitted.
Response: 82 09 06. (This response indicates the message was actually transmitted.)
8x 09 06 jj kk ll mm (time stamp is appended to the end, if enabled).
(Time stamp is two or four bytes, depending on model.)

Short Form Transmit Command

The AVT-418 and AVT-718 units, firmware version 3.0 and above, offer a 'short form' transmit command.

To use this command, the user must have selected the baud rate and physical layer prior to issuing the transmit command. No other setup to transmit is required.

An example of the 'short form' transmit command is provided here.

⇒ 06 06 03 C4 1A 2B 3C --> *82 09 06*
Command explanation:

06:	Header byte; 0 - to the network; 6 - six bytes follow.
06:	0 - 11-bit arbitration field; 6 - object number.
03 C4:	Arbitration field.
1A 2B 3C:	Data field, 3 bytes.

Response explanation:

82:	8 - command response; 2 - two bytes follow.
09:	Transmit status report, object transmission successful.
06:	Object number.

A Receive Example

The following example demonstrates setting up an object to receive a message.
Object #12 is used here.

This example is not related to the previous example or the demonstration.

All commands are shown. Responses are in *italics*.

⇒ Set up the Global Message Masks, as required. (Default status is recommended.)

11-bit mask:	73 01 xx yy --> *83 01 xx yy*
29-bit mask:	75 02 rr ss tt vv --> *85 02 rr ss tt vv*

⇒ Disable the object (if necessary).
73 04 0C 00 --> *83 04 0C 00*

⇒ Set the object configuration.
79 05 0C 01 10 05 16 B7 C8 D9 --> *89 05 0C 01 10 05 16 B7 C8 D9*
Explanation:

79 05:	Command (upper nibble 7), byte count (lower nibble 9), and command type (05).
0C:	Object number.
01:	Receive.
10:	29-bit arbitration field.
00:	Data length (not used).
16 B7 C8 D9:	Arbitration field.

⇒ Enable the object.
73 04 0C 01 --> *83 04 0C 01*

⇒ Notification that a message has been received; 11-bit arbitration field.
Response: 0x 0C aa bb cc rr ss tt vv xx.
Explanation:

0x:	Message from the bus (upper nibble 0). Byte count to follow (lower nibble x).
0C:	Object number.
aa bb cc:	ID of received message; 11-bit arbitration field.
rr ss tt vv xx:	Received data.

⇒ Notification that a message has been received; 29-bit arbitration field.
Response: 0x 8C aa bb cc dd rr ss tt vv xx.
Explanation:

0x:	Message from the bus (upper nibble 0). Byte count to follow (lower nibble x).
8C:	Object number is “C.” [“8” indicates a 29-bit arbitration field].
aa bb cc dd:	ID of received message; 29-bit arbitration field.
rr ss tt vv xx:	Received data.

⇒ Note that the alternate header format of 11 xx ... will be used if the byte count exceeds $0F.

A Receive Everything Example

The following example demonstrates setting up object #15 to be able to receive every (11-bit ID) message on the network. (If other objects are set up as receive, object #15 will receive all messages NOT received by any other receive objects.

Object #15 is selected to receive all network messages since it cannot be set as a transmit object. In order to receive all messages on the network the message mask must be set to all zeros. If any objects are set as transmit, the Global Mask (7x 01 or 7x 02 commands) should **NEVER** be set to all zeros. This could result in a transmit object repeatedly transmitting a message without delay and without stopping (the so called "babbling idiot" failure). Refer to the Intel 82527 document (referenced on the next page) for a detailed explanation of how this could happen.

This example is not related to the previous demonstration.

All commands are shown. Responses are in *italics*.

⇒ Select CAN mode of operation.
E1 99 --> *91 10*

⇒ Select the baud rate:
72 0A xx --> *82 0A xx*
[where xx is the baud rate according to the list of available baud rates as listed in the "Master Commands and Responses" document.]

⇒ Select the physical layer: CAN-C, CAN-B, or SWC.
72 11 xx --> *82 11 xx*
[where xx is the mode selection according to the list of modes in the "Master Commands and Responses" document.]

⇒ Set object #15 message mask to all zeros.
75 03 00 00 00 00 --> *85 03 00 00 00 00*

⇒ Set up object #15.
79 05 0F 01 zz 00 00 00 00 00 --> *89 05 0F 01 zz 00 00 00 00 00*
79 - header byte (7x command, 9 bytes follow).
05 - command type.
0F - object number
01 - receive
zz - arbitration field length;
zz = 01 for 11-bit arbitration field
zz = 10 for 29-bit arbitration field.
00 - data length [this is not important]
00 00 00 00 - arbitration field value

⇒ Enable object #15
73 04 0F 01 --> *83 04 0F 01*

At this point all network messages will be received by object #15 and sent to the host.

Note: The AVT interface never sends to the host computer any message it transmits.

A Keep Alive Example

Problem: You need to send the module under test a “Keep Alive” message every second. You also need to send the module specific messages during the testing. All of these messages use the same message ID (or arbitration field). How to do this ??

Wrong Method

- Declare object #1 as a transmit object and use it to transmit your test messages.
- Declare object #2 as a transmit object. Load it with the keep alive message. Set object #2 for Type0 Periodic message.

This will not work - because:

- You must never set up and have enabled more than one transmit object with the same message ID (arbitration field) at the same time. It is possible and likely that you will turn the AVT interface unit into a “babbling idiot”. [I won’t go into the details here of why. If you don’t understand, write or call me and I’ll explain.]

Solution #1

- Handle both the keep alive and test messages in your application software.
- This will work but could be cumbersome and/or difficult to implement.

Solution #2 - A better method

- Set up Type1 Periodic Message #1 (which uses object #1) as your keep alive message.
- Send your test messages out object #1 using the “short form” transmit command.
- The AVT-718 code will automatically prevent either message from being lost or corrupted.

Solution #2 - Example

```
1. F1 A5                          ; reset the interface unit
2. E1 99                          ; switch to CAN mode
3. 72 0A 04                       ; 125 kbps
4. 72 11 02                       ; two wire CAN-C
5. 77 18 01 10 01 00 07 44        ; Type1 Periodic Message #1 setup
6. 76 19 01 68 6A F1 3F           ; Type1 Periodic Message #1 data
7. 72 1E 01                       ; Type1 timer interval 10 msec.
8. 74 1B 01 00 64                 ; Type1 Periodic Message #1 to 100 = 1000 msec.
9. 73 1A 01 01                    ; Type1 Periodic Message #1 enabled
```

The AVT unit will send the programmed message using object #1 once per second until reset or disabled.

Send your test messages using object #1 and the short form of the transmit command.

```
0B 01 07 44 01 02 03 04 05 06 07 08      ; test message with dummy data of:  01 - 08
```

Caution

Do not use the 7x 06 and 7x 07 commands to send your test messages. It will not work properly.

ISO 15765 Processing

ISO 15765 is a detailed standard for the formatting of the data field of a CAN message. It permits the transmission of any length of data from 00 to 4095 bytes (even though a CAN frame limits the data to 8 bytes). It also permits the addition of an "address extension" byte. ISO 15765 message format processing is, for the most part, independent of arbitration field length (message ID), baud rate, or physical layer.

Messages that use the ISO 15765 messaging format may be referred to as: "Segmented Messages" or "Multi-Frame Messages" or by other names.

AVT-418 and AVT-718 firmware version 5.1 and above incorporate ISO 15765 messaging support. The user does not need to keep track of data length or frame types to send or receive messages. The firmware handles all functions involved in transmitting or receiving ISO 15765 formatted messages.

The term: "ISO 15765" may be referred to as: "I5P" here.

Command Summary - Enable/Disable

The ISO 15765 function is enabled by the user with the 7x 26 or 7x 28 commands.

The 7x 28 command was first available in firmware version 5.9.
It is the recommended command to enable I5P functionality.

The 7x 26 command function was implemented in early firmware versions.
It is not recommended for new designs.

Only use one command to enable ISO 15765 processing.
Do not use both commands.

Do not enable any Auto Respond Objects if ISO 15765 processing is enabled.

Object #15 ($0F) does NOT support ISO 15765 processing.

7x 28 Command - Recommended

ISO 15765 Processing Command

71 28:	Object pairing status query. Default is no objects are paired.
72 28 0x:	Disable ISO 15765 processing for object 0x and it's pair.
73 28 0x 0y	Declare objects 0x and 0y as paired. Enable ISO 15765 processing for the pair. Address extension is disabled for the pair. (Object order is not important.)
74 28 0x 0y ww	Declare objects 0x and 0y as paired. Enable ISO 15765 processing for the pair. Enable address extension and use byte ww in flow control frame.

Response

82 28 00:	No objects are paired.
83 28 0x 0y:	Objects 0x and 0y are paired. ISO 15765 processing is enabled for the pair. Address extension is disabled for the pair.
84 28 0x 0y ww:	Objects 0x and 0y are paired. ISO 15765 processing is enabled for the pair. Address extension is enabled for the pair.

If the address extension feature of ISO 15765 is used, the user must specify the address extension byte for the flow control frames. Our experience has shown that the majority of CAN networks using ISO 15765 do not use the address extension function. However, some GM-LAN networks may use it.

7x 26 Command

The 7x 26 command is not recommended for new designs.

ISO 15765 Processing Command

71 26:	ISO 15765 support status query.
72 26 00:	ISO 15765 support disabled. {Default.}
73 26 01 0x:	ISO 15765 support enabled. Normal addressing. Flow control transmit object is: 0x [01 to 0E].
74 26 02 0x yy:	ISO 15765 support enabled. Extended addressing. Flow control transmit object is: 0x [01 to 0E]. Flow control address extension byte is: yy.

Object Pairing

When ISO 15765 processing is enabled using the 7x 26 command, all receive messages are subjected to ISO 15765 processing. There is only one transmit object associated with all the received messages. This is not adequate when communicating with multiple modules using different message IDs. There may also be instances where some network messages conform to ISO 15765 and others do not.

The 7x 28 command was developed to address the shortcomings of the other command.

The 7x 28 command is used to declare an object pair. When an object pair is declared, ISO 15765 processing is enabled for the pair of declared objects only. All other objects are not affected.

It is possible to declare multiple receive objects and pair them with the same transmit object.
For example, the user may set up three receive objects and one transmit object. Each of the receive objects can be paired with the one transmit object. This is done by issuing three 7x 28 commands; one for each of the receive objects and using the same transmit object in each command.

Message Padding

Some CAN networks require that all CAN frames be full length. This is done by padding the data field with extra bytes, if necessary, to create a full frame. With I5P enabled, the user can enable or disable frame padding. The 7x 27 command is used to enable frame padding and the user can specify the value of the pad byte. The AVT interface will then pad the data field, if necessary. The AVT interface will not confuse the data bytes from the pad bytes - that is a function of ISO 15765 processing.

The 7x 27 command disables or enables the message padding feature, only when the ISO 15765 function is enabled. That command is:

71 27:	ISO 15765 message padding status query.
72 27 00:	Disable ISO 15765 message padding.
72 27 01:	Enable ISO 15765 message padding. {Default} All data fields are 8 bytes long. Default pad byte value is 00.
73 27 01 xx:	Enable ISO 15765 message padding. "xx" = pad byte value.

Using ISO 15765 Processing

- Enable ISO 15765 processing after entering CAN mode.
- Setup and enable both the transmit and receive objects of the pair.
- No action is required to receive an ISO 15765 message from the network, regardless of length.
- The AVT interface will remove the "PCI" control byte and any pad bytes from a received message.
- If enabled, the AVT interface will leave the address extension byte in the data field.
- Do not include the "PCI" control byte in the data field of a message to be transmitted.
- Do not include any pad bytes in the data field of a message to be transmitted.
- No special action is required to send a message, regardless of length.
- To transmit a message to the network, the user must use the short form transmit command. Data lengths from 00 to 4095 bytes are supported. Normal or extended addressing is supported.
- There are a range of transmit commands available to the user. A complete list is provided in the "Master Commands and Responses" document.
- If address extension is enabled, include the AE byte as the first byte of the data field.
- Periodic messages transmitted by the AVT unit (both Type0 and Type1) are not subjected to ISO 15765 processing. The data field of those messages must be properly configured by the user.

Example Setup

A number of assumptions have been made for this example. A real application may be different.

⇒ Select CAN mode of operation.
E1 99

⇒ Select the baud rate:
72 0A xx

⇒ Setup and enable object #3 for receive.
77 05 03 01 01 00 ww xx
73 04 03 01

⇒ Setup and enable object #6 for transmit operations.
[Necessary if a flow control frame must be transmitted.]
77 05 06 10 01 00 yy zz
73 04 06 10

⇒ Enable ISO 15765 message processing.
Objects #3 and #6 are paired; object #6 is the designated transmit object; normal addressing is used.
73 28 03 06

⇒ Select the physical layer: CAN-C, CAN-B, or SWC.
72 11 xx

Note the order. The physical layer is selected last. This prevents receiving any messages from the network before all the setup has been completed - which might cause errors or unexpected operations.

The AVT unit will monitor all received messages and process them according to ISO 15765 rules.

A received message will, generally, be of the form:

0x 0y jj kk aa bb cc ... :

0x - count of bytes to follow.

0y - object number (1 to F).

jj kk - 11-bit arbitration field, right justified.

aa bb cc ... : data from object.

A complete list of received message formats is in the “Master Commands and Responses” document.

<u>Example #1</u>

An ISO 15765 message is received from the network:

07 03 ww xx 11 22 33 44

07 - indicates from the network, 7 bytes follow.

03 - message received from object #3.

ww xx - ID of the message.

11 22 33 44 - data field.

The AVT unit removed the “PCI” byte (value of 04) and removed all pad bytes, if any.

<u>Example #2</u>

Transmit the following message onto the network, using object #6:

Message ID: yy zz

Data field: 12 34 56 78 90

Send the following command to the AVT unit.

08 06 yy zz 12 34 56 78 90

Receive the following response, when the message has been successfully transmitted:

82 09 06

The AVT unit checks the length of the data field, computes the “PCI” byte, constructs the CAN frame, adds pad bytes (if enabled), and transmits the frame or frames onto the network.

<u>Example #3</u>

Transmit the following message onto the network:

Message ID: yy zz

Data field: 12 34 56 78 90 AB CD EF 11 12 13 14 15 16 17 18

Send the following command to the AVT unit.

11 13 06 yy zz 12 34 56 78 90 AB CD EF 11 12 13 14 15 16 17 18

Receive the following response, when the message has been successfully transmitted:

```
82 09 06
```

The data field is more than 7 bytes. The AVT unit will handle the multi-frame messaging required.

Additional Information

A separate document was prepared for several customers about ISO 15765 processing for
AVT-418 and AVT-718 firmware version 5.9 and above. The contents of that document follow.
It is pretty much the same information as provided above, but it may be presented in a different manner and different issues were of concern.

Questions ??
Write or call for clarification and guidance.

The most up-to-date version of the "Master Commands and Responses" document is available for downloading from our web site at:
http://www.AVT-HQ.com/download.htm#Notes

Works Cited

Intel. 82527 Serial Communications Controller Architectural Overview *Automotive*.
Mt. Prospect, IL: Intel Corp., February 1995. Order number: 272410-002.

Note: "I5P" = ISO 15765 processing

CAN mode operations of AVT-418/718 firmware versions 5.1 to 5.8 had ISO 15765 processing [I5P] available. There were several operational limitations to that function.

- The operator could only specify a single transmit object for flow control frames.
- All transmit and receive objects were enabled or disabled globally.

With increasingly complex applications it is necessary to:

- Be able to specify a transmit object for each receive object (object pairing).
- Be able to enable or disable I5P processing for individual objects.

AVT-418/718 firmware versions 5.9 and above provide the capability to enable or disable I5P processing by object number and declare object pairs.

Statement: ISO 15765 processing necessarily requires the declaration of object pairs.

Technically this is not true, but the AVT interface does not have 'a-priora' information about the size of any message (to or from the network). It must be assumed that multi-frame messages will exist. That requires every object (to or from the network) to have an associated opposite function object (from or to the network). Hence, an object pair.

One object of the pair must be configured to receive. The other object must be configured to transmit. The user must set up the transmit object with the correct message ID.

For completeness the "Address Extension" (AE) byte exists in all commands.
The declared AE byte is only used in transmitting a flow control frame.

The new command to support these enhanced operations is the "Pairing" command: 7x 28.
A description of the pairing command follows.

Some "sequence of event" listings are provided to illustrate how the host computer might use this new function and command.

NOTE: AVT-418 and AVT-718 firmware is identical.

I5P Pairing Command

It is implicit in this command that if an object pair is declared, messages through both of the objects are processed according to the ISO 15765 protocol. If an object is not paired, then it is processed 'normally' - ISO 15765 processing is disabled for that object.

The order of the pairing is not important. (Object 0x may be the receive or transmit object.)

One object of the pair must be configured to receive. The other object must be configured to transmit. The transmit object must be set up with the proper message ID. The user is responsible for properly setting up and enabling each object of the pair. This must be done before issuing the pairing command.

Commands and Responses

• 71 28	Report all object pairs	
	Response if none exist:	82 28 00
	Response if any exist:	83 28 0x 0y
	or:	84 28 0x 0y AE
	All objects are reported if any exist.	
• 72 28 00	Delete all object pairs	
	Response:	82 28 00
• 72 28 0x	Delete one object pair	
	Response:	82 28 0x
• 73 28 0x 0y	Configure objects 0x and 0y as a pair; AE is disabled	
	Response:	83 28 0x 0y
• 74 28 0x 0y AE	Configure objects 0x and 0y as a pair. Use AE for any response frames	
	Response:	84 28 0x 0y AE
	[Object order is not important.]	

Sequences of Events

Initial Setup

- Enter CAN mode [E1 99]
- Select baud rate [72 0A xx]
- Select physical layer [72 11 xx]
- *** Operations commence from this point.

Initialize, Enable, and Declare an object pair

- Determine receive object [0r] to be used and the message ID for that object.
- Determine the transmit object [0t] to be used and message ID for that object.
- Determine if AE is used
  if not - ignore this
  if so - determine AE byte value
- Set up the receive object.
  [7x 05 0r 01 ...]
- Set up the transmit object.
  [7x 05 0t 10 ...]
- Declare the object pair.
  [7x 28 0r 0t ...] or [7x 28 0t 0r ...]
- Enable the transmit object [73 04 0t 10]
- Enable the receive object [73 04 0r 01]
- *** All messages from the network with the specified ID will be received, I5P processed, and sent to the host at this time.
- *** All messages to be sent to the network must use the ‘short form’ transmit command. They will be processed, formatted, and transmitted on to the network according to ISO 15765.
- *** Note: The order of the commands. The receive object is enabled last.

Disable a receive message ID

- Determine the receive object number [0r]
- Disable the receive object [73 04 0r 00]
  (strongly recommended)
- Disable the transmit object [73 04 0r 00]
  (optional, but recommended)
- Disable I5P for that object pair [72 28 0r]

Transmit a message onto the network

- Depending on how many data bytes are to be transmitted, there are three forms of the 'short form' transmit command available.
- Include the address extension [AE] byte, if used.
- Do not include the 'PCI' byte.
- Do not include any pad bytes.

⇒ Use the 'short form' transmit command.

```
     0x tt mm nn ae pp qq ...
               0x       = count of number of bytes that follow.
               tt       = transmit object to use.
               mm nn    = message ID.
               Ae       = optional address extension byte.
               pp qq ... = actual data bytes of the message.
```

⇒ or

```
     11 xx tt mm nn ae pp qq ...
               xx       = count of number of bytes that follow.
               tt       = transmit object to use.
               mm nn    = message ID.
               Ae       = optional address extension byte.
               pp qq ... = actual data bytes of the message.
```

⇒ or

```
     12 0y xx tt mm nn pp qq ...
               xx yy    = count of number of bytes that follow.
               tt       = transmit object to use.
               mm nn    = message ID.
               Ae       = optional address extension byte.
               pp qq ... = actual data bytes of the message.
```

- *** The AVT-418/718 will handle all the necessary operations to properly send the message onto the network using the ISO 15765 protocol.

Receiving a message from the network

- Depending on how many data bytes were received, the AVT-418/718 will inform the host using one of three possible forms. The general form is described here. Details are provided below.
- Total count of bytes that follow.
  [0x or 11 xx or 12 xx yy; one, two, or three bytes]

- The object number the message was received through.
  (If 29-bit ID was used, bit #7 is set.) [one byte]
- The message ID.
  [one byte]
- AE byte, if used.
  [one byte]
- All of the message data.
- PCI bytes are omitted.
- Pad bytes are omitted.

Examples

⇒ 0x tt mm nn ae pp qq ...

0x	= count of number of bytes that follow.
rr	= object number message was received through.
mm nn	= message ID.
Ae	= optional address extension byte.
pp qq ...	= actual data bytes of the message.

⇒ or

11 xx tt mm nn ae pp qq ...

xx	= count of number of bytes that follow.
rr	= object number message was received through.
mm nn	= message ID.
Ae	= optional address extension byte.
pp qq ...	= actual data bytes of the message.

⇒ or

12 0y xx tt mm nn pp qq ...

xx yy	= count of number of bytes that follow.
rr	= object number message was received through.
mm nn	= message ID.
Ae	= optional address extension byte.
pp qq ...	= actual data bytes of the message.

- *** The AVT-418/718 handles all the necessary operations to properly receive the message from the network using ISO 15765 protocol.

Error Trapping

The user is strongly recommended to include in their application software a means or method for trapping, and preferably storing, all error codes received from the AVT-418/718 unit. All error messages from the AVT-418/718 are of the form:

2x yy zz ...

Examples:

22 55 02: indicates that a received message had an incorrect PCI byte.

25 55 36 xx yy zz: indicates that a buffer timed out.
xx = buffer number.
yy = mode byte.
zz = flag byte.

A complete list of the error codes is provided below and in the “Master Commands and Responses” document.

Contact this office for clarification and explanation if error codes are received.

Summary - Paired Mode (version 5.9 and above)

- I5P is enabled/disabled by object pair.
- AE is enabled/disabled by object pair.
- Receive object uses paired transmit object to transmit flow control frame.
- Transmit object uses paired receive object to receive flow control frame.
- Specified AE byte is used only in transmitting a flow control frame.
- User must set up receive object.
- User must set up transmit object.
- Transmit short form command should be used.
  Specified object number and ID must be consistent with setup.
- Transmit long form method bypasses I5P processing.
- Periodic messages transmitted by the AVT unit (Type0 or Type1) are not I5P processed.

Pairing Command

71 28:	Object pairing status query. Default is no objects are paired.
72 28 0x:	Disable ISO 15765 processing for object 0x and it’s pair.
73 28 0x 0y	Declare objects 0x and 0y as paired. Enable ISO 15765 processing for the pair. Address extension is disabled for the pair. (Object order is not important.)
74 28 0x 0y ww	Declare objects 0x and 0y as paired. Enable ISO 15765 processing for the pair. Enable address extension and use byte ww in flow control frame.

Pairing Command Responses

82 28 00:	No objects are paired.
83 28 0x 0y:	Objects 0x and 0y are paired. ISO 15765 processing is enabled for the pair. Address extension is disabled for the pair.
84 28 0x 0y ww:	Objects 0x and 0y are paired. ISO 15765 processing is enabled for the pair. Address extension is enabled for the pair.

Error codes

2x 55: ISO 15765 processing error. Error code follows.

01:	Receive mode; buffer time out, buffer number follows.
02:	Receive mode; PCI frame type error
03:	Receive mode; PCI byte count greater than or equal to CFG byte count.
04:	Receive mode; first frame received, no small buffers available.
05:	[Receive mode; flow control frame transmit attempt time out.]
06:	Receive mode; flow control frame, data field too short.

07: Receive mode; consecutive frame, buffer assigned but not in-use.
08: Receive mode; consecutive frame, sequence number error.
09: Receive mode; flow control frame, separation time invalid.
0A: Receive mode; DLC > 7 in single frame processing.
0B: Receive mode; DLC > 6 in single frame processing.
0C: Receive mode; DLC > 6 in first frame processing.
0D: Receive mode; DLC = 0.
0E: Receive mode; DLC = 1 with extended addressing enabled.
0F: Time out forwarding message to host.
10: Receive mode; FFDL = 0.
11: Transmit mode; buffer time out.
12: Transmit mode; error decoding op mode flags.
13: Transmit mode; buffer underrun on first frame.
Buffer and object numbers follow.
14: Transmit mode; invalid flow status received, operation aborted.
Buffer and object numbers follow.
15: Transmit mode; flow status = 2 received, operation aborted.
Buffer and object numbers follow.
16: Transmit mode; flow status = 1, wait count max, operation aborted.
Buffer and object numbers follow.
17: Transmit mode; buffer count = 0 on first frame, operation aborted.
Buffer and object numbers follow.
18: Buffer manager; buffer mode value wrong.
19: Receive mode; first frame DLC too short.
1A: Received a flow control frame, not expecting one.
20: [Buffer time out; op flag bit 0 set; long message first frame waiting.]
21: [Buffer time out; op flag bit 1 set; waiting for flow control frame.]
22: [Buffer time out; op flag bit 2 set; consecutive frames are in-progress.]
23: [Buffer time out; op flag bit 3 set; flow control frame received.]
24: [Buffer time out; op flag bit 4 set; flow control frame waiting to be loaded.]
25: [Buffer time out; op flag bit 5 set; transmit object triggered.]
26: [Buffer time out; op flag bit 6 set; not used.]
27: [Buffer time out; op flag bit 7 set; not used.]
28: [Buffer time out; no op flags found.]
29: Receive mode; first frame received, large buffer not available.
2A: Consecutive frame; buffer not assigned.
2B: Consecutive frame; buffer number not valid.
2C: Flow control frame; buffer not assigned.
2D: Flow control frame; buffer number not valid.
2E: Buffer service counter failure; zero.
2F: Buffer service counter failure; not valid.
30: Transmit service counter failure; zero.
31: Transmit service counter failure; not valid.
32: Time out flushing TX1 command.
33: TX1 command pending timed out.
34: TX0 command pending timed out.
35: Time out flushing 1X command.
36: Buffer time out; buffer number, mode byte, and flag byte follow.

Summary - Legacy mode (version 5.8 and below).

- I5P is enabled/disabled globally.
- AE is enabled/disabled globally.
- All objects are paired to only one object.
- The designated object is used to send a flow control frame.
- A received flow control frame is not associated to a specific transmit object.
- Only one buffer available; used for both transmit and receive functions.
- User must set up all receive objects.
- User should set up designated transmit object.
- Transmit short form commands are I5P processed.
- Transmit long form method bypasses I5P processing.
- A transmit ack (82 09 0x) will be sent to the host when the AVT unit transmits a flow control frame. This was an error. The AVT unit should not have done this. Version 5.9 and above do not send the host this message.

SOF	PDU 11 - bits	SRR	IDE	PDU 18 - bits	RTR	r1	r0	DLC 4 - bits	Data Field 0 to 64 bits	CRC 16 - bits	ACK Field 2 - bits	EOF 7 - bits

J1939 Frame (CAN 2.0B) Frame Construction

PDU (Protocol Data Unit) construction (29 - bits)

[each "x" = one bit]

Priority	rsvd.	data page	Format (1 byte)	Destin. addr. (1 byte)	Source addr. (1 byte)
x x x	0	x	x x x x x x x x	x x x x x x x x	x x x x x x x x

PGN (Parameter Group Number) construction (24 - bits)

[each "x" = one bit]

not used	rsvd.	data page	Format (1 byte)	Destin. addr. (1 byte)
x x x x x x	0	x	x x x x x x x x	x x x x x x x x