Introduction

Almost all automobiles manufactured after 1996 provide an interface from which test equipment can obtain diagnostic information. The data transfer on these interfaces follow several standards, The OBDPro interface is designed to act as a bridge between these On-Board Diagnostics (OBD) ports and standard PC RS232 ports. The interface can automatically sense and convert the nine most common protocols in use today.

Key Features

- Supports 9 OBDII protocols
- Automatically searches for a protocol
- Fully configurable with AT commands
- Configurable RS232 speeds (9600 to 128000 bps)
- Voltage input for battery monitoring

Overview

The following sections will describe the detailed operation of the OBDPro interface. There are a number of PC based clients in the market that will work seamlessly with this interface. A list of some of the most commonly used products is given in the appendix.

These clients talk to the Interface using simple text commands which means that you could talk to the chip by just typing in the commands at a terminal program connected to the interface via a Serial cable. The AT commands implemented by the OBDPro have been chosen so as to make the chip compatible with similar chips available on the market such as ELM327 and AGV OBD interface

Communicating with the OBDPro interface

The OBDPro interface uses a standard RS232 type serial connection to communicate with the user. The data rate when the unit is shipped defaults to 9600 bauds, 8 databits, no parity and 1 stop bit, but the data rates are customizable using the ATSCS command.

The OBDPro interface has two 9 pin serial connectors, (The USB version has a 9 pin serial for the OBD Cable and a USB connector for the PC) the end labeled "**car**" plugs into the black OBD cable whereas the other end labeled "**pc**" plugs into the computer via a straight through serial cable. When the OBDPro interface is powered up, the four LED's will blink (the pattern will depend on the current baud rate – refer to ATSCS and ATWCS commands in the datasheet)and will then send the message

OBDPro v1.0

>

This indicates that the computer connections and terminal software settings are correct. The '>' character displayed above is the prompt character indicating that the device is ready to receive commands over the RS232 port. Messages sent on the serial port can either be intended for the interface's internal use, or for reformatting and passing on to the OBD bus. The interface determines where the received characters are to be directed by analyzing the

http://www.obdpros.com

Note: Portions of the datasheet; specifically the sections "OBD commands" and general OBD discussions on setting headers and monitoring messages have been adapted from the ELM 327 datasheet available at www.elmelectronics.com for use with their products.

entire string once the complete message has been received. Commands for the OBDPro's internal use will always begin with the characters 'AT' (similar to modem commands), while commands for the OBD bus can only contain the ASCII codes for hexadecimal digits (0 to 9 and A to F). All messages to the OBD Prompt must be terminated with a carriage return character (hex '0D') before it will be acted upon. If a command has been started but not terminated by a '0D' it will be aborted after 20 seconds of inactivity and the interface will print a question mark ('?') to indicate an error in entering the command.

Messages that are not understood by the OBDPro (syntax errors) will always be indicated by a single question mark "?". These include incomplete messages, incorrect AT commands, or invalid hexadecimal digit strings. When sending OBD II commands meant for the vehicle the OBDPro interface does not check for validity of the message. The interface only checks to ensure that an even number of hex digits were received, but cannot check if a command would be considered valid by the vehicle. (When a command is not recognized by the vehicle it will not return any data and the interface will send back a NO DATA string).
When sending commands; users should always wait for the prompt character ('>') before sending the next command.

Couple of notes on command syntax:

- The interface is not case-sensitive, so typing an OBD command such as ABCD is equivalent to saying abCd
- Space characters and all control characters (tab, linefeed, etc.) in the input are ignored, so they can be inserted anywhere to improve readability.
- Sending a single carriage return character by itself will repeat the last command; this comes in handy when one needs to continue requesting the same data from the vehicle. E.g. dynamically changing data such as engine RPM.

AT Commands

Introduction:

Several parameters within the OBDPro interface can be adjusted in order to modify its behavior. These do not normally have to be changed before attempting to talk to the vehicle, but offer flexibility in using the interface. For example by turning off the character echo mode or adjusting a timeout value a slight increase in speed might be achieved.

In some instances the user might want to change the header bytes to address a specific controller within the vehicle. In order to do this, internal 'AT' commands must be issued.

The OBDPro constantly monitors the data sent by the PC, looking for messages that begin with the character 'A' followed by the character 'T'. If found; the next characters will be interpreted as internal configuration or 'AT' commands, and will be executed upon receipt of a terminating carriage return character.

The OBDPro will reply with the characters 'OK' on the successful completion of a command, so the user knows that it has been executed. Some of the commands allow numbers as arguments to set the internal values. These will always be hexadecimal numbers which must generally be provided in pairs. For the on/off types of commands, the second character is the number 1 or 0, 1 meaning "on" and 0 meaning "off"

ATBI [Bypass Initialization sequence]

This command allows an OBD protocol to be made active without requiring any initialization or handshaking. Use caution while using this command as some vehicles (mainly ISO protocols) will not respond to OBD commands without initialization.

ATCF hhh [set the CAN ID Filter to hhh]

Tha CAN filter works along with the CAN mask to determine the vehicle messages that should be displayed. As the OBDPro receives each message, the incoming CAN ID bits are compared to the CAN Filter bits (when the mask bit is a ‘1’). If all of the relevant bits match, the message will be accepted, and processed by the interface, otherwise it will be discarded. This three nibble version of the CAN Filter command is a convenient way to set the 11 bit ID on 11 bit CAN systems. Only the rightmost 11 bits of the provided nibbles are used, and the most significant bit is ignored.

The Can filter and masks can be set at any time even when the Can protocol is not active, if the filters and masks are set incorrectly the Scantool will typically respond with “NO DATA” upon sending an OBD command. In this case you can restore the original filters and masks by issuing the ATZ command.

ATCF hh hh hh hh [set the CAN ID Filter to hhhhhhhh]

This command allows all 29 bits of the CAN Filter to be set at once. The 3 most significant bits will always be ignored, and can be given any value. This command can also be used to enter the 11 bit ID filters, in which case you would need to issue the command as ATCF 00 00 0h hh.

CFC0 and **CFC1** [CAN Flow Control off or on]

The ISO 15765-4 protocol expects a “Flow Control” message to always be sent in response to a “First Frame” message. The interface automatically sends these. The AT CFC0 command prevents sending of the flow control message. The default setting is CFC1 - Flow Controls on.

Note that during monitoring (ATMA, ATMR, or ATMT), no Flow Control messages are sent regardless of the CFC option setting.

ATCM hhh [set the CAN ID Mask to hhh]

Since the CAN bus is extremely busy; at any given time the user would be interested in only a subset of the messages being transmitted, instructing the OBDPro interface to only look at messages with specific CAN ID’s is accomplished by the filter, which works in conjunction with the mask. A mask is a group of bits that indicate which bits in the filter are relevant, and which ones can be ignored. A ‘must match’ condition is indicated by setting a mask bit to '1', while a 'don't care' condition is indicated by setting a bit to '0'. Similar to te Can filter command this three digit variation of the ATCM command is used to provide mask values for 11 bit ID systems (the most significant bit is always ignored).

ATCM hh hh hh hh [set the CAN ID Mask to hhhhhhhh]

This command is used to assign mask values for 29 bit ID systems. You could also assign the 11 bit CAN mask using this command just use leading zero’s except for the last three nibbles so the command would be

Note: The three most significant bits that you provide in the first digit will be ignored.

ATCP hh [set CAN Priority bits to hh]

This command is used to set the five most significant bits in a 29 bit CAN ID word (the other 24 bits are set with the ATSH command). The default value for these priority bits is hex 18.

ATCV dddd [Calibrate the Voltage to dd.dd volts]

The interface has a built in voltmeter to measure the battery voltage of the vehicle it's hooked up to. The voltage reading shown by the interface on receiving the ATRV command can be calibrated with this command. The argument ('dddd') must always be provided as 4 digits, with no decimal point (it assumes that a decimal place is between the second and the third digits).

To use this calibration feature, use an accurate voltmeter to read the actual input voltage. If, for example, the interface consistently says the voltage is 12.6V when you measure 12.2 volts, simply issue AT CV 1220, and the device will recalibrate itself and subsequent ATRV commands will display 12.2 volts. If you use a test voltage that is less than 10 volts, remember to add a leading zero (that is, 8.1 volts should be entered as ATCV 0810).

ATD [set all to Defaults]
This command is used to set the options to the default (or factory) settings, Any settings that the user had made for custom headers, filters, or masks will be restored to their default values, and all timer settings will also be restored to their defaults.

ATDP [Describe the current Protocol]

The OBDPro interface automatically determines the appropriate OBD protocol to use for each vehicle that it is connected to. When the interface connects to a vehicle, however, it returns only the data requested, and does not report the protocol found. The ATDP command is used to display the current protocol that the OBDPro Interface is set to. If the automatic option is also selected, the protocol will show the word "AUTO" before it, followed by the protocol.

Note that the actual protocol names are displayed, not the numbers used by the protocol setting commands.

ATDPN [Describe the Protocol by Number]

This command is similar to the ATDP command, but it returns a number which represents the current protocol. If the automatic search function is also enabled, the number will be preceded with the letter 'A'. The number is the same one that is used with the ATSP (set protocol) command

ATE0 and **ATE1** [Echo off (0) or on(1)]

These commands control whether or not characters received on the serial port are re-transmitted (or echoed) back to the host computer. To reduce traffic on the RS232 bus, you can turn echo off by issuing ATE0. The default is echo on.

ATH0 and **ATH1** [Headers off (0) or on(1)]

All the OBD protocol messages have a header section which is normally not shown by the interface but can be by issuing the ATH1 command. Turning the headers on actually shows more than just the header bytes - you will see the complete message as transmitted, including the check-digits,and PCI bytes.

In the case of CAN protocol currently we do not display the CAN data length code (DLC), the CAN CRC.

Also in case of J1850 PWM, the IFR byte used to acknowledge messages is not shown.

ATI [Identify yourself]
Issuing this command causes the interface to print the startup product ID string (currently "ELM327 v1.0a compatible –

OBDPro v1.0"). The PC software uses this to determine which interface it is talking to.

ATL0 and **ATL1** [Linefeeds off (0) or on(1)]

If the ATL1 is issued, linefeeds will be generated after every carriage return character, and for ATL0, it will be off. Typically leave linefeeds on (the default) if you are using a terminal program

To reduce the communications overhead, OBD Software packages should turn linefeeds off. The interface always defaults to linefeeds on when initially powered up.

ATMA [Monitor All messages]

This command puts the OBDPro interface into a bus monitoring mode where it displays all messages that is sees on the OBD bus. This continues indefinitely until stopped by activity on the Serial input. To stop the monitoring, send a single character and wait for the interface to respond with a prompt character ('>').

The character used to abort the monitoring will be discarded and will not be interpreted as part of the next command.

ATMR hh [Monitor for Receiver hh]

This command puts the OBDPro interface into a bus monitoring mode, where it only displays messages that were sent to the hex address given by hh. These are messages that have the value hh in the second byte of a traditional three byte OBD header, in bits 8 to 15 of a 29 bit CAN ID, or in bits 8 to 10 of an 11 bit CAN ID. Any single RS232 character aborts the monitoring.

ATMT hh [Monitor for Transmitter hh]

This command puts the OBDPro interface into a bus monitoring mode where it only displays messages sent by transmitter address hh. These are messages that found have the hh value in the third byte of a traditional three byte OBD header, or in bits 0 to 7 for CAN systems. Any RS232 activity (single character) aborts the monitoring.

ATR0 and **ATR1** [Responses off (0) or on(1)]

These commands control the OBDPro's automatic display of responses. If responses have been turned off, the OBDPro will not wait for a reply from the vehicle after sending a request, and will return immediately to wait for the next RS232 command. The default is R1, or responses on.

ATRV [Read the input Voltage]

This command initiates reading the voltage powering the OBDPro interface. The interface is capable of measuring a voltage from 4.5 volts up to about 20V, with an uncalibrated accuracy of about 2%. Please refer to the ATCV command for details on calibrating the reported voltage.

ATSCS h [Set Communications Speed]

This command allows the user to switch the current RS-232 baud rate to one of the values listed below.

h	Speed
1	9600
2	14400
3	19200
4	38400
5	56600
6	115200
7	128000

The Baud rate can be saved using the ATWCS command. The OBDPro interface will respond with "OK" at the old speed, displays the prompt character ('>'), and then switch to the new speed. The factory default

speed if 9600 baud to ensure compatibility with existing software.

If the new communications speed is not saved via the ATWCS command the OBDPro will revert to the old baud rate on the next reset.

When the OBDPro is powered up the front led's will indicate the communication speed that the interface is set to for two seconds.

	PC			CAR	
Speed	**TX**	**RX**	**PWR**	**TX**	**RX**
9600	ON	OFF	ON	OFF	OFF
14400	OFF	ON	ON	OFF	OFF
19200	OFF	OFF	ON	ON	OFF
38400	OFF	OFF	ON	OFF	ON
56600	ON	ON	ON	OFF	OFF
115200	ON	ON	ON	ON	OFF
128000	ON	ON	ON	ON	ON

ATSH xx yy zz [Set the Header to xx yy zz]

This command enables the user to set their own header bytes. The header bytes are normally assigned values based on the current protocol (and are not required to be adjusted), but when you want to address manufacturer specific modules using physical addressing you may need to change the default values. The value of hex digits xx will be used for the first or priority/type byte, yy will be used for the second or receiver/target byte, and zz will be used for the third or transmitter/source byte.

These remain in effect until set again, or until restored to their default values with the ATD, or ATZ commands. This command is used to assign all header bytes, whether they are for a J1850, ISO 9141, ISO 14230, or a CAN system. The CAN systems will use these three bytes to fill bits 0 to 23 of the ID word (for a 29 bit ID), or will use only the rightmost 11 bits for an 11 bit CAN ID. The additional 5 bits needed for a 29 bit system are provided through the ATCP command (since they rarely change).

ATSH xyz [Set the Header to 00 0x yz]

Entering an 11 bit CAN ID word (header) requires that extra leading zeros be added (eg. AT SH 00 07 DF). This command provides a simpler way to set the header. The SH xyz AT command accepts a three digit argument, takes only the right-most 11 bits from that, adds leading zeros, and stores the result in the header storage locations for you. As an example, AT SH 7DF would set the CAN ID to hex 7DF which is used by 11 bit CAN OBD.

ATSP h [Set Protocol to h]

This command is used to force the interface to use a particular protocol when sending/receiving messages from the vehicle, issuing this command also makes this protocol the default.

Currently, the valid protocols are:
0 - Automatic
1 - SAE J1850 PWM (41.6 Kbaud)
2 - SAE J1850 VPW (10.4 Kbaud)
3 - ISO 9141-2 (5 baud init, 10.4 Kbaud)
4 - ISO 14230-4 KWP (5 baud init, 10.4 Kbaud)
5 - ISO 14230-4 KWP (fast init, 10.4 Kbaud)
6 - ISO 15765-4 CAN (11 bit ID, 500 Kbaud)
7 - ISO 15765-4 CAN (29 bit ID, 500 Kbaud)
8 - ISO 15765-4 CAN (11 bit ID, 250 Kbaud)
9 - ISO 15765-4 CAN (29 bit ID, 250 Kbaud)

The Automatic selection (protocol 0) instructs OBDPro to try all protocols, when looking for a valid one. It will first try protocol 1, then will sequence through each of the others, until the correct one used by the vehicle is identified.

If a non zero protocol is selected with this command (eg. ATSP 3), that protocol will become the default. Failure to initiate a connection in this situation will result in responses such as BUS INIT: ...ERROR. If you know the protocol used by your vehicle;

setting the protocol manually saves time in trying to detect the protocol.

ATSP Ah [Set Protocol to Auto, h]

This is a variation of the ATSP command and allows you to set a starting (default) protocol, while retaining the ability to automatically search for a valid protocol on a failure to connect. For example, if your vehicle is J1850 VPW, but you want to occasionally use the OBDPro scantool on other vehicles, you would issue command AT SP A2. The default protocol will then be 2, but with the ability to automatically search for other protocols.

ATST hh [Set Timeout to hh]

After sending a request, the OBDPro waits a preset time before declaring that there was no response from the vehicle (the 'NO DATA' response). Even if there was a response, the OBDPro will wait this time to be sure that there are no more responses coming. The ATST timeout setting controls the amount of time that the interface waits.

The actual time that the interface waits is 4 msec x hh, so passing a value of FF would result in a maximum time out of just over one second. A value of 00 sets the timeout to the default value of 200 ms.

Note: The 200 ms is an extremely conservative number some of the protocols specify a much shorter timeout for responses, you can speed up the response times by experimenting with reduced timeout values.

ATSW hh [Set Wakeup to hh]

For the ISO 9141 & ISO 14230 protocols; once a data connection is made with a vehicle, there needs to be a data flow every few seconds or the connection will terminate. The OBDPro automatically generates keep alive messages to keep the communications link "alive"

The time interval between these periodic 'wakeup' messages can be adjusted in 20 msec increments using the AT SW hh command, where hh is any hexadecimal value from 00 to FF.

A value of FF (decimal 255) results in the maximum possible delay of just over 5 seconds. The default setting provides a nominal delay of 3 seconds between messages. The value 00 will stop all periodic messages. This should be used with caution as it could result in a NO_DATA message if the vehicle terminates the connection.

ATWCS [Write Communications Speed]

This command saves the current baud rate in nonvolatile memory.

To set the default speed first change the current speed (Please see the ATSCS command), make sure communication is possible at this new data rate, then issue the ATWS command.

Example:

```
>ATSCS 4
OK
>AT
OK
>ATWCS
OK
>
```

In the example above, we're switching to 38,400 baud speed, making sure that we can communicate with the interface by issuing the AT command and reading the response, and then writing the new data rate to nonvolatile memory with the ATWS command.

On subsequent power ups the interface will always use the 38,400 baud rate.

ATWM xx yy zz aa or
ATWM xx yy zz aa bb or

ATWM xx yy zz aa bb cc [set Wakeup Message to…]

These commands allow the user to override the default settings for the wakeup messages. The user must provide the three header bytes (xx yy zz), and either one (aa), two (aa bb) or three data bytes (aa bb cc). The checksum byte is automatically calculated by the interface and should not be provided.

The message provided will be periodically sent at the rate determined by the AT SW setting. The replies to this message are not printed by the interface.

Byte values assigned with this command are not affected by those set with other commands (AT SH) and do not have any effect on the transmission of normal OBD request messages.

This command only applies for the ISO 9141 and ISO 14230 protocols.

Z [reset all]

This command causes the OBDPro to perform a complete reset as if power were cycled off and then on again. All settings are returned to their default values, and the OBDPro will be put in the idle state, waiting for characters on the RS232 bus.

AT Command Summary

General Commands

D set all to Defaults
E0 Echo Off
E1 Echo On
I print the ID
L0 Linefeeds Off (default set by pin 7)
L1 Linefeeds On
SCS Set Communication speed
WCS Write comm. speed
Z reset all

OBD Commands

BI Bypass the Initialization sequence
DP Describe the current Protocol
DPN Describe the Protocol by Number
H0 Headers Off (default)
H1 Headers On
MA Monitor All
MR hh Monitor for Receiver = hh
MT hh Monitor for Transmitter = hh
R0 Responses Off
R1 Responses On
SH yzz Set Header
SH xx yy zz Set Header
SP h Set Protocol to h
SP Ah Set Protocol to Auto, h and save it
ST hh Set Timeout to hh x 4 msec

CAN Specific Commands

CF hhh set the ID Filter to hhh
CF hh hh hh hh set the ID Filter to hhhhhhhh
CFC1 CAN Flow Control On
CFC0 CAN Flow Control Off
CM hhh set the ID Mask to hhh
CM hh hh hh hh set the ID Mask to hhhhhhhh
CP hh set CAN Priority (only for 29 bit)

ISO Specific Commands

SW hh Set Wakeup interval to hh x 20 msec

WM xx yy zz aa set the Wakeup Message
WM xx yy zz aa bb " "
WM xx yy zz aa bb cc " "

Misc. Commands

CV dddd Calibrate the Voltage to dd.dd volts
RV Read the Voltage

AT Command examples:

Lets look at how we would send some sample AT commands to the interface. Once you plug in the interface to the OBD connector in the vehicle and hookup the serial connection type "ATI" and hit enter.

The interface will respond with "OBDPro v1.0" or the custom version string that you have instructed it to use.

Here is a screen representation of the command.

>ATI
ELM327 1.0a compatible OBDPro v1.0

Next we will change the communication speed

Type in atscs 4, the interface should respond with OK and you should no longer be able to type any characters (Since your terminal was setup at the default baud rate of 9600 and we just changed the baud rate to 38,400). Configure your terminal to use the higher speed of 38,400 and then type in ati you should see the "OBDPro v1.0" string displayed at the new baud rate. In order to save the baud rate for subsequent sessions issue atwcs and the scantool will respond with "OK"

Note that we did not use upper case characters in this example, the OBDPro interface will accept upper case (ATI) as well as lower case (ati) or any combination of these (Ati).

Also note that although we type in atscs 4 this is only to separate the commands and make them more readable. You do not have to add spaces, or if you wish, you can add many spaces – it does not affect the internal interpretation of the command.

Other AT Commands are used in the same manner. Simply type the letters A and T, follow by the command you want to send, then any arguments that are required for that command, and press Enter.

OBD Commands

Any command sent to the OBDPro interface that does not begin with "AT" is assumed to be meant for the vehicle. Each pair of ASCII bytes is validated to ensure that they are hexadecimal digits, and will then be combined into single data bytes for transmitting to the vehicle. OBD commands are actually sent to the vehicle embedded in a data packet.

The OBDPro interface will normally limit the number of bytes that can be sent to seven (14 hexadecimal digits); the maximum number allowed by the standards. Attempts to send either an odd number of hex digits, or too many digits will result in an error – the entire command is then ignored and a single question mark printed.

As an example of sending a command to the vehicle, assume that 01 00 is the command that is required to be sent. In this case, the user would type "01 00" and hit Enter. The OBDPro interface would store the characters as they are received, and when the carriage return was received, it would begin to assess the command, convert it to a two byte value of 1 followed by a 0 add the header bytes and a checksum byte and then send the resulting total of 6 bytes to the vehicle.

Note: The carriage return character is only a signal to the interface, and is not sent on to the vehicle. After sending the command, the interface listens on the OBD bus for messages, looking for ones that are directed to it. If a message address matches, those received bytes will be sent over the serial port, while messages received that do not have matching addresses will be ignored. Sometimes a single command send to the vehicle would result in multiple ECU's

reporting back, in order to allow for this the OBDPro interface will wait for the timeout specified by the ATST commandf before it stops listening to the OBD bus.

Querying the vehicle for OBD data

The OBD II standards specify that each group of bytes sent to the vehicle must adhere to a set format. The first byte ('mode' byte) describes the type of data being requested, while subsequent bytes specify the actual information required (given by a 'parameter identification' or PID number). The modes and PIDs are described in detail in the SAE document J1979 (ISO 15031-5).

The 9 diagnostic modes specified by J1979 are
01 - show current data
02 - show freeze frame data
03 - show diagnostic trouble codes
04 - clear trouble codes and stored values
05 - test results, oxygen sensors
06 - test results, non-continuously monitored
07 - show "pending" trouble codes
08 - special control mode
09 - request vehicle information

Within each mode, PID 00 is normally reserved to show which PIDs are supported by that mode.

Mode 01, PID 00 must be supported by all vehicles, and can be accessed as follows: Once the OBDPro interface is properly connected to your vehicle, and powered, with the ignition turned on, issue the mode 01 PID 00 command:

```
>01 00
```

You might see a bus initialization message or a protocol search message and then the response from the vehicle; shown below is a representative response.

```
41 00 BF BF B9 93
```

The 41 00 signifies a response (4) from a mode 1 request from PID 00 (a mode 2, PID 00 request is answered with a 42 00, etc.). The next four bytes (BF, BF, B9, and 93) represent the requested data, in this case a bit pattern showing the PIDs supported by this mode (1=supported, 0=not).

Now lets query the vehicle for the coolant temperature, this is PID 05 in mode 01, and can be requested as follows:

```
>01 05
```

The response will be of the form:

```
41 05 3A
```

The 41 05 shows that this is a response to a mode 1 request for PID 05, while the 3A is the desired data. Converting the hexadecimal 3A to decimal, we get 3 x 16 + 10 = 58. This represents the current temperature in degrees Celsius, but with a zero offset to allow for subzero temperatures. To convert to the actual coolant temperature, you need to subtract 40 from the value obtained. So the actual coolant temperature is 58 - 40 or 18 °C.

Multiline Responses

There are occasions when a vehicle must respond with more information than one 'message' is able to show. In these cases, it responds with several lines which must be assembled into one complete message. One example of this is a request for the vehicle id number (VIN) of the vehicle (mode 09, PID 02). This is a multi line response that needs to be assembled. One example from a SAE J1850 PWM vehicle is shown below

```
>0902
49 02 01 31 46 41 46
49 02 02 50 33 36 33
49 02 03 38 59 57 32
49 02 04 39 31 31 31
49 02 05 33 00 00 00
```

Note that all OBD compliant vehicles do not provide this information. If your vehicle does not support this parameter, you will only see a “NO DATA” response. The first two bytes (49 and 02) on each line of the above response do not show any vehicle information. They only show that this is a response to a 09 02 request. Assembling the remainder of the data and ignoring the trailing zeroes gives:

```
31 46 41 46 50 33 36 33 38 59
57 32 39 31 31 31 33
```

Converting these hex digits to ASCII gives the following serial number for the vehicle:

```
1 F A F P 3 6 3 8 Y W 2 9 1 1 1
3
```

CAN systems will display this information in a format shown below

```
>0902
014
0: 49 02 01 59 56 31
1: 4D 53 33 38 32 32 36
2: 32 31 36 36 35 36 33
```

CAN Formatting has been left on (the default), making the reading of the data easier. With formatting on, the sequence numbers are shown with a colon (‘:’) after each, so that they clearly stand out (0:, 1:, etc.). CAN systems add this hex digit (it goes from 0 to F then repeats), to aid in reassembling the data, just as the J1850 vehicle did. The first line of this response says that there are 014 bytes of information to follow. That is 14 in hexadecimal, or 20 in decimal terms, which agrees with the 6 + 7 + 7 bytes shown on the three lines. Serial numbers are generally 17 digits long however, so how do we assemble the number from 20 digits? The second line shown begins with the familiar 49 02, as this is a response to a 09 02 request. Clearly they are not part of the serial number. CAN will occasionally add a third byte to the response which we see next (‘01’) showing the number of data items in the response (the vehicle can only have one VIN, so the response says there is only one data item). That third byte can be ignored. This leaves 17 data bytes which are the serial number. All that is needed is a conversion to ASCII in order to read them, exactly as before.

A final example shows a different type of multi line response that can occur when two or more ECU’s all respond to one request. The following is a typical response to a 01 00 request:

```
>0100
41 00 80 10 80 01
41 00 BF BE B9 93
```

```
To find out which ECU is
sending the data we need to
turn on headers
```

```
>at h1
OK
>0100
7EA 06 41 00 80 00 00 01 AA
7E8 06 41 00 BF BF B9 93 AA
```

Now, if you analyze the header, you can see that the third byte shows ECU 10 (the engine controller) and ECU 18 (the transmission) both responding. Usually the multi line responses are relatively straight-forward to decipher, but they do take some practice.

Setting the Headers

The emissions related diagnostic trouble codes as described in the SAE J1979 standard (ISO15031-5) represent only a portion of the data that a vehicle may have available – much more can be obtained if you are able to direct the requests elsewhere.

Example: In a GM vehicle the class 2 messages are used to even command the body controller to execute a door unlock etc.

Accessing the OBDII diagnostics information requires that requests be made to what is known as a 'functional address.' Any processor that supports the function will respond to the request (and theoretically, many different processors can respond to a single functional request). Every processor (or ECU) will also respond to what is known as their physical address. It is the physical address that uniquely identifies each module in a vehicle, and permits you to direct more specific queries to only one particular module.

To retrieve information beyond that of the OBDII requirements then, it will be necessary to direct your requests to either a different functional address, or to an ECU's physical address. This is done by changing the data in the message header. As an example of functional addressing, let us assume that you want to request that the processor responsible for Engine Coolant provide the current Fluid Temperature. You do not know its address, so you consult the SAE J2178 standard and determine that Engine Coolant is functional address 48. J2178 also tells you that for your J1850 VPW vehicle, a priority byte of A8 is appropriate. Then, knowing that a scan tool is normally address F1, you form the information into the three header bytes of A8 48 and F1.

To tell the ObdPro to use these new header bytes, all that is needed is the Set Header command:

Some requests, being of a low priority, may not be answered immediately, possibly causing a "NO DATA" result. Such instances might require a longer timeout period to get the response.
If you do not know the address, recall that the sender of information is usually shown in the third header byte. By monitoring your system for a time with the headers turned on (AT H1), you can quickly learn the main addresses of the senders. When you know the address, simply use it for the second byte in the header. Physical addressing is used by standards such as SAE J2190 to provide a great deal of vehicle information. Many of the details of how to access this information (the PID numbers, etc.) is proprietary and manufacturers rarely share the info.

ISO14230 standard also specifies that the first header byte must always include the length of the data field. The interface always determines the number of bytes that you are sending, and inserts that length automatically, so you only need to provide the two format bits.

Addressing within the CAN (ISO 15765-4) protocols is quite similar in many ways. First, consider the 29 bit standard. The OBDPro splits the 29 bits into a CAN Priority byte and the three header bytes that we are now familiar with. Figure 1 below shows how these are combined for use by the OBDPro interface.

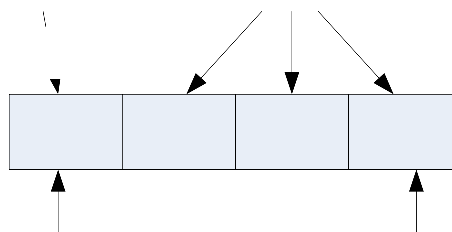

Figure 1 Setting the 29 bit CAN ID

The CAN standard states that for diagnostics, the priority byte ('vv' in the diagram) shall always be **1B**. Using a separate instruction to set these 'priority' bits should be only a minor inconvenience, as they are rarely changed. The next byte ('xx') describes the type of message that this is, and is set to hex DB for functional addressing, and to DA if using physical addressing. The next two bytes are as defined previously

for the other standards - 'yy' is the receiver (or Target Address), and 'zz' is the transmitter (or Source Address). For the functional diagnostic requests, the receiver is always 33, and the transmitter is F1.

Now let's discuss the header used in 11 bit CAN systems. They also use a priority/address structure, but shorten it into roughly three nibbles rather than three bytes. The OBDPro uses the same commands to set these values as for other headers, except that it only uses the 11 least significant bits of the provided header bytes, and ignores the others (as shown in Figure 2).

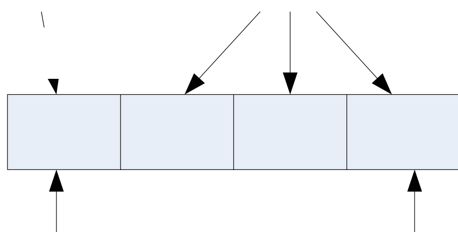

Figure 2 Setting the 11 bit CAN ID

It quickly becomes inconvenient to have to enter six digits when only three are required, so there is a special 'short' version of the AT SH command that accepts three hex digits. It actually operates by simply adding zeroes for you.

The 11 bit CAN standard typically makes functional requests (ID/header = 7DF), but receives physical replies (7En). With headers turned on, it is a simple matter to learn the address of the module that is replying, then use that information to make physical requests if desired. For example, if the headers are on, and you send 01 00, you might see:

```
>0100
7E8 06 41 00 BF BF B9 93 AA
```

The 7E8 shows that ECU#1 has responded. In order to talk directly to that ECU, all you need is to set the header to the appropriate value (it is 7E0 – see ISO 15765-4 for more information). From that point on, you can 'talk' directly to the ECU using its physical address:

```
>atsh 7e0
>0100
7E8 06 41 00 BF BE B9 93 38
>0105
7E8 03 41 05 89 BE B9 93 38
```

Of course, it's a little confusing seeing the headers at all times, so you may want to turn them off again.

Monitoring the Bus

Some vehicles use the OBD bus for information transfer during normal vehicle operation, passing a great deal of information over it. To see how your vehicle uses the OBD bus, you can enter the OBDPro's 'Monitor All' mode, by sending the command AT MA from your terminal program.

Once received, the interface will continually display information that it sees on the OBD bus, regardless of transmitter or receiver addresses. Note that the periodic 'wakeup' messages are not sent while in this mode, so if you have an ISO 9141 or ISO 14230 bus that had been initialized previously, it may 'go to sleep' while monitoring. The monitoring mode can be stopped by sending any single RS323 character to the interface.

The interface will start monitoring the bus using the currently active protocol. In some vehicles more than one protocol might be used. Eq – For newer GM cars the OBD messages use the CAN protocol but some of the body electronics continue to use the Class II protocol (J1850 VPW), in this case if the interface was used for getting diagnostics data from the vehicle the interface will be set to the the CAN bus and will monitor the vehicle in this

mode also. If you want to monitor data on the Class II bus the interface should be switched to the J1850 VPW protocol by issuing an ATSP 2 command.

If the "Monitor All' command provides too much information (specifically for the CAN bus), then you can restrict the range of data that is to be displayed. If we are only interested in seeing messages that are being transmitted by the ECU with address 10. we could restrict the messages by typing

```
>atmt 10
```

All messages that contain 10 in the third byte of the header will be displayed.

This command also applies to the 11 bit CAN systems the CAN ID is actually stored as the least significant 11 bits in the 3 byte 'header storage' location. It will be stored with 3 bits in the receiver's address location, and the remaining 8 bits in the transmitter's address location. For this example, we have requested that all messages created by transmitter '10' be printed, so all 11 bit CAN IDs that end in 10 will be displayed (ie 'x10').

The AT MR command operates in a similar fashion, but now it looks for messages from being sent to a specific address For example, to use it to look for messages being sent to the ECU with address 10 send

```
>AT MR 10
```

and all messages that contain 10 in the second byte of the header will be displayed. For 11 bit CAN ID's this the interface only looks at the least significant three bits in the message To search for all CAN messages that begin with a 2, then you will need to use the command 'AT MR 02', and to see all of the 7xx's, you will need to use 'AT MR 07'.

Monitoring Example:

Here we will discuss one example on how you would use the ATMA command to discover vehicle communication messages. In some vehicles these messages are used to control a number of accessories such as the windows and locks. One such vehicle is the 2006 GM Trailblazer, where the commands to open the windows and lock/unlock doors is sent over the J1850 VPW bus.

Let's investigate how to discover the message used to open the windows. In order to uncover what messages are sent when you hit the power window button, we need to first set the OBDPro Scantool to monitor the J1850 VPW protocol.

Connect the OBDPro Scantool and issue the command ATSP2. This forces the OBDPro into using the J1850 VPW protocol. Since we would like to also identify the required header for sending the window open message, we will also issue the show headers command ATH1. Now Issue the ATMA command to start monitoring the bus.

You will see a bunch of messages being sent by various modules as shown below

```
>atsp2
OK
>ath1
OK
>atma
C8 53 11 30 00 86
08 FF 97 03 68
08 FF 80 03 25
08 FF 89 03 33
88 83 11 0A 34 CC 33
88 25 29 07 00 97
08 FF 29 03 8F
08 FF 98 03 CB
08 FF 60 03 73
C8 53 11 30 00 86
```

Now hit the "open window" button and carefully watch the messages scroll by on the terminal you will see a bunch of messages similar to this

```
68 CB A0 09 08 11 00 00 70 E2
```

```
68 CB A0 09 02 11 00 00 70 41
C8 53 11 30 00 86
68 CB A0 09 08 11 00 00 70 E2
68 CB A0 09 01 11 00 00 70 FF
```

Repeatedly monitoring the messages reveals that
`68 CB A0 09 02 11 00 00 70 41`
Might be the message for activating the windows, as a further test lets monitor messages that are meant for receiver address 0xCB by issuing ATMR CB. Once we issue this command we notice that we only see messages when we hit the window down or window up button in the car. This confirms the window down message. Now let's issue the window down message from the OBDPro Scantool.

To do this we need to change the header, we accomplish this by issuing the ATSH command
ATSH 68 CB A0.

Then issue the command "09 02 11 00 00 70" and you should see the front passenger window roll down.

CAN Message filtering

The monitoring commands (AT MA, MR and MT) usually work very well with the 'slower' protocols – J1850, ISO 9141 and ISO 14230. But they are not sufficient for the CAN bus since there is a lot more information passing over them. To help reduce the amount of information seen by the interface, it has a 'filter' that can be used to pass only messages with specific ID bits. A range of values can be passed when the filter is used with a 'mask' to say which bits are relevant.

Example, consider an application where you are trying to monitor for 29 bit CAN diagnostic messages, exactly like the OBDPro does. By definition, these messages will be sent to the scan tool at address F1. From ISO 15765-4, you know then that the ID portion of the reply must be of the form: 18 DA F1 xx where xx is the address of the module that is sending the message. To use the filter, then, enter what you have into it, putting anything in for the unknown portion (you will see why in a moment). The command to set the CAN filter is AT CF…
`>AT CF 18 DA F1 00`
How do you tell the OBDPro to ignore those last two 0's? You do that with the mask. The mask is a set of bits that tell the OBDPro which bits in the filter are relevant. If the mask bit is 1, that filter bit is relevant, and is required to match. If it is 0, then that filter bit will be ignored. All bits in the above message are relevant, except those of the last two digits. To set the mask for this example then, you would need to use the CAN Mask command, as follows:
`>AT CM 1F FF FF 00`

If desired, you can convert the hexadecimal to binary to see what has been done. The 11 bit CAN IDs are treated in the same manner. Recall that they are stored internally in the right-most 11 bits of the locations used for 29 bit CAN, which must be considered when creating a filter or mask. As an example, assume that we wish to display all messages that have a 6 as the first digit of the 11 bit ID.

We need to set a filter to look for 6 in that digit:
`>AT CF 00 00 06 00`
The 11 bit ID is stored in the last three locations, so the 6 would appear where it is shown. Now, to make that digit relevant, we create the mask:
`>AT CM 00 00 0F 00`

The system only uses the 11 right-most bits in this case, so we can be lazy and enter the F as shown (the first bit of the F will be ignored, and it will be treated as if we had entered a 7). Clearly, this can be quite cumbersome if using 11 bit CAN systems routinely. To help with that, the OBDPro offers some shorter versions of

the CF and CM commands. You need only enter:

```
>AT CF 600
```

and

```
>AT CM F00
```

for the above example. The commands work internally by simply entering the extra 00's for you. As for the full eight digit versions, only the 11 least significant (rightmost) digits are used, so you do not need to take special care with the first bit. With a little practice, these commands are fairly easy to master. Initially, try entering the filter and mask values, then use a command such as AT MA to see what the results are. The OBDPro knows that you are trying to filter, and combines the effects of both commands (it will do that for MR and MT as well). The MA, MR and MT commands also have the extra benefit that if they are in effect, the OBDPro will remain quiet, not sending acknowledgement or error signals, so anything you do while monitoring should not disrupt others that are on the bus. Note that if a filter has been set, it will be used for all CAN messages, so standard OBD requests may then respond with "NO DATA". If you are having trouble, reset everything to the default values.

CAN Message Formats

The ISO 15765-4 standard defines several message types that are to be used with diagnostic systems. Currently, there are four main ones that are used:

SF - the Single Frame
FF - the First Frame (of a multiframe message)
CF - the Consecutive Frame (of a multiframe message)
FC - the Flow Control frame

The Single Frame message contains storage for up to seven data bytes and what is known as a PCI (Protocol Control Information) byte. The PCI byte is always the first byte of them all, and tells how many data bytes are to follow.

A First Frame message is used to say that a multiframe message is about to be sent, and tells the receiver just how many data bytes to expect. The length descriptor is limited to 12 bits, so a maximum of 4095 byes can be received at once using this method. Consecutive Frame messages are sent after the First Frame message to provide the remainder of the data. Each Consecutive Frame message includes a single hex digit 'sequence number' that is used to help with reassembling the data. It is expected that if a message were corrupted and resent, it could be out of order by a few packets, but not by more than 16. As seen previously, the serial number for a vehicle is often a multi frame response:

```
>0902
014
0: 49 02 01 31 47 4E
1: 45 53 31 33 4D 35 36
2: 32 32 31 38 32 33 34
```

In this example, the line that begins with 0: is the First Frame message. The length (014) was actually extracted from the message by the OBDPro scantool and printed on the separate line as shown.

Following the First Frame line are two Consecutive Frames as shown (1: and 2:). To learn more details of the exact formatting, you may want to send a request such as the one above, then repeat the same request with the headers enabled (AT H1). This will show the PCI bytes that are actually used to send these components of the total message.

The Flow Control frame is one that you do not normally have to deal with. When a First Frame message is sent as part of a reply, the OBDPro must tell the sender some technical things such as how long to delay between Consecutive Frames, etc.

These are predefined by the ISO 15765-4 standard and are not alterable by the user. The only thing that you can do with them is to disable the sending of Flow Control messages entirely (AT CFC0). This may be required if experimenting with a different CAN system.

Bus Initialization

Both the ISO 9141-2 and ISO 14230-4 (KWP2000) standards require that the vehicle's OBD bus be initialized before any communications can take place. The ISO 9141 standard allows for only a slow (2 to 3 second) process, while ISO 14230 allows for both the slow method, and a faster alternative. In either case, once the bus has been initiated, communications must take place at least once every five seconds, or the bus will revert to a low-power 'sleep' mode. The OBDPro takes care of this bus initiation and the periodic sending of 'keep-alive' or 'wakeup' messages for you – it is automatic and requires no input from the user. The OBDPro will not perform the bus initiation until the first message needs to be sent, however. During the automatic search process, you will not see any status reporting while the initiation process is taking place, but if you have the Auto option off, then you will see a message similar to this:

BUS INIT: ...

The three dots appear only as the slow initiation process is carried out - a fast initiation does not show them. This will be followed by either the expression 'OK' to say it was successful, or else an error message to indicate that there was a problem. (The most common error encountered is in forgetting to turn the vehicles key to 'ON' before attempting to talk to the vehicle.)

Once initiated, the OBDPro does what is required to keep the bus alive, without any intervention from the user. The automatic messages being sent every few seconds will be visible via the blinking OBD activity LED's.

Wakeup Messages

After an ISO 9141 or ISO 14230 connection has been established, periodic activity on the bus need to occur in order to maintain that connection. If normal requests and responses are being sent, that is usually sufficient, but occasionally filler messages need to be sent to prevent the connection from timing out. These periodic messages are called 'Wakeup Messages.' They keep the connection alive, and prevent the circuitry from going back to the idle or sleep mode.

OBDPro automatically creates and sends these for you if there appears to be no other activity there is nothing that you need do to ensure that they occur. To see these, once a connection is made, simply monitor the OBD transmit LED - you will see the periodic binks created when the OBDPro send a wakeup message.

The standards state that if there is no activity at least every five seconds, the connection may close. To ensure that this does not happen, by default the OBDPro will send a wakeup message after three seconds of inactivity. This time interval is fully programmable, should you prefer something different (see the AT SW command).

The user has the capability to change the wakeup message. To do this, refer to the AT WM message format.

Error Messages

BUFFER FULL

The OBDPro provides a 256 byte internal RS232 transmit buffer so that OBD messages can be received quickly, stored, and sent to the computer at a more constant rate. Occasionally (particularly with CAN systems) the buffer will fill at a faster rate than it is being 'emptied.' Eventually it becomes full, and no more data can be stored (it is lost).
If you are receiving BUFFER FULL messages, and are using a 9600 baud data rate, try changing your data rate to 38400 baud. If you still receive BUFFER FULL messages after that, consider some of the filtering options (the MR, MT, CF and CM AT Commands).

BUS BUSY

The OBDPro interface tried to send the mode command or initialize the bus, but detected too much activity to insert a message. This could be because the bus was in fact busy, but may be due a wiring problem that is giving a continuously active input.

BUS ERROR

A generic problem occured. This is most often from an invalid signal being detected (a long pulse, etc.) on the bus, or a wiring error.

CAN ERROR

The CAN system had difficulty initializing, sending, or receiving. Often this is simply from not being connected to a CAN system when you attempt to send a message.

DATA ERROR

There was a response from the vehicle, but the information was incorrect or could not be recovered.

<DATA ERROR

The was an error in the line pointed to, either from an incorrect checksum or a problem with the format of the message (the OBDPro still displays the data received). There could have been a noise burst which interfered, or a circuit problem. Try re-sending the command.

NO DATA

The OBDPro interface waited for the period of time that was set by AT ST, but detected no response from the vehicle. It may be that the vehicle had no data to offer, that the mode requested was not supported, that the vehicle was attending to higher priority issues, or in the case of the CAN systems, the filter may have been set to ignore the response. Try adjusting the AT ST time to be sure that you have allowed sufficient time to obtain a response, or restoring the CAN filter to its default setting.

<RX ERROR

An error was detected in the received CAN data. This will usually only occur if monitoring a CAN bus, while set for an incorrect baud rate. Try a different protocol.

UNABLE TO CONNECT

The OBDPro tried all of the available protocols, and could not detect a compatible one. This could be because your vehicle uses an unsupported protocol, or could be as simple as forgetting to turn the ignition key on. Check all of your connections, and the ignition, then try the command again.

?

This is the standard response for a misunderstood command received on the RS232 input. Usually it is due to a typing mistake.