ELM327

OBD to RS232 Interpreter

Description

Almost all new automobiles produced today are required, by law, to provide an interface from which test equipment can obtain diagnostic information. The data transfer on these interfaces follow several standards, none of which are directly compatible with PCs or PDAs. The ELM327 is designed to act as a bridge between these On-Board Diagnostics (OBD) ports and a standard RS232 interface.

The ELM327 builds on improved versions of our proven ELM320, ELM322, and ELM323 interfaces by adding seven CAN protocols to them. The result is an IC that can automatically sense and convert the most common protocols in use today. There are a number of other improvements as well – a high speed RS232 option, battery voltage monitoring, and customizable features through programmable parameters, to name only a few.

The ELM327 requires few external components to make a fully functioning circuit. The following pages discuss the interface details, and show how to use the IC to 'talk' to your vehicle, then concludes with two schematics to get you started.

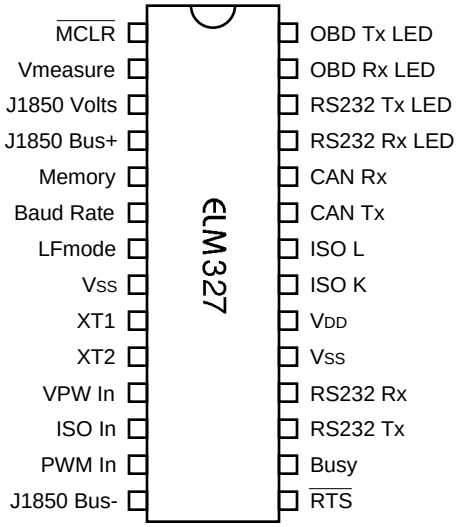
Applications

- Diagnostic trouble code readers
- Automotive scan tools
- Teaching aids

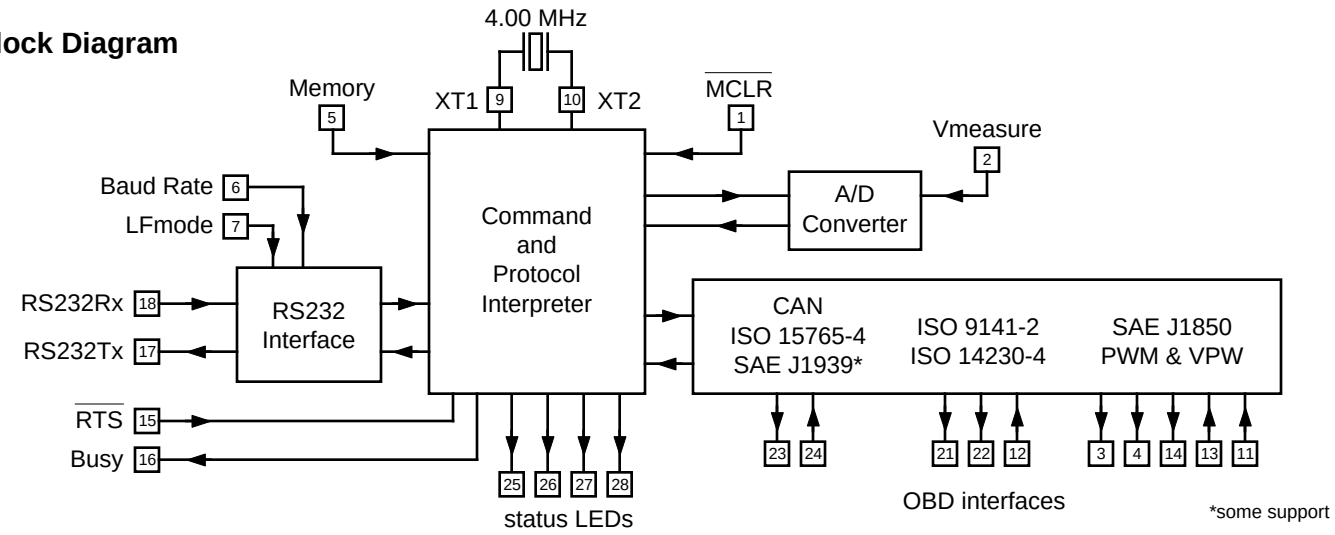
Features

- Supports 12 protocols
- RS232 baud rates to 500Kbps
- Automatically searches for protocols
- Fully configurable with AT commands
- Voltage input for battery monitoring
- Low power CMOS design

Connection Diagram
PDIP and SOIC
(top view)



Block Diagram



Pin Descriptions

$\overline{\text{MCLR}}$ (pin 1)

A momentary (>2µsec) logic low applied to this input will reset the ELM327. If unused, this pin should be connected to a logic high (V_{DD}) level.

Vmeasure (pin 2)

This analog input is used to measure a 0 to 5V signal that is applied to it. Care must be taken to prevent the voltage from going outside of the supply levels of the ELM327, or damage may occur. If it is not used, this pin should be tied to either V_{DD} or V_{SS}.

J1850 Volts (pin 3)

This output can be used to control a voltage supply for the J1850 Bus+ output. The pin will output a logic high level when a nominal 8V is required (for J1850 VPW), and will output a low level when 5V is needed (as for J1850 PWM applications). If this switching capability is not required for your application, this output can be left open-circuited.

J1850 Bus+ (pin 4)

This active high output is used to drive the J1850 Bus+ Line to an active level. Note that this signal does not have to be used for the Bus- Line (as was the case for the ELM320), since a separate J1850 Bus- drive output is provided on pin 14.

Memory (pin 5)

This input controls the default state of the memory option. If this pin is at a high level during power-up or reset, the memory function will be enabled by default. If it is at a low level, then the default will be to have it disabled. Memory can always be enabled or disabled with the AT M1 and AT M0 commands.

Baud Rate (pin 6)

This input controls the baud rate of the RS232 interface. If it is at a high level during power-up or reset, the baud rate will be set to 38400 (or the rate that has been set by PP 0C). If at a low level, the baud rate will always be 9600.

LFmode (pin 7)

This input is used to select the default linefeed mode to be used after a power-up or system reset. If it is at a high level, then by default messages sent by the ELM327 will be terminated with both a carriage return and a linefeed character. If it is at a low level, lines will be terminated by a carriage return only. This behaviour can always be modified by issuing an AT L1 or AT L0 command.

V_{SS} (pins 8 and 19)

Circuit common must be connected to these pins.

XT1 (pin 9) and XT2 (pin 10)

A 4.000 MHz oscillator crystal is connected between these two pins. Loading capacitors as required by the crystal (typically 27pF each) will also need to be connected between each of these pins and circuit common (Vss).

Note that this device has not been configured for operation with an external oscillator – it expects a crystal to be connected to these pins. Use of an external clock source is not recommended.

VPW In (pin 11)

This is the active high input for the J1850 VPW data signal. When at rest (bus recessive) this pin should be at a low logic level. This input has Schmitt trigger waveshaping, so no special amplification is required.

ISO In (pin 12)

This is the active low input for the ISO 9141 and ISO 14230 data signal. It is derived from the K Line, and should be at a high logic level when at

All rights reserved. Copyright 2005, 2006, 2007, and 2008 by Elm Electronics Inc.
Every effort is made to verify the accuracy of information provided in this document, but no representation or warranty can be given and no liability assumed by Elm Electronics with respect to the accuracy and/or use of any products or information described in this document. Elm Electronics will not be responsible for any patent infringements arising from the use of these products or information, and does not authorize or warrant the use of any Elm Electronics product in life support devices and/or systems. Elm Electronics reserves the right to make changes to the device(s) described in this document in order to improve reliability, function, or design.

Pin Descriptions (continued)

rest (bus recessive). No special amplification is required, as this input has Schmitt trigger waveshaping.

PWM In (pin 13)

This is the active low input for the J1850 PWM data signal. It should normally be at a high level when at rest (ie. bus recessive). This input has Schmitt trigger waveshaping, so no special amplification is required.

J1850 Bus- (pin 14)

This active high output is used to drive the J1850 Bus- Line to an active (dominant) level for J1850 PWM applications. If unused, this output can be left open-circuited.

$\overline{\text{RTS}}$ (pin 15)

This active low “Request To Send” input can be used to interrupt the OBD processing in order to send a new command. Normally high, the line is brought low for attention, and should remain so until the Busy line (pin 16) indicates that the ELM327 is no longer busy. This input has Schmitt trigger waveshaping.

Busy (pin 16)

This active high output shows the current state of the ELM327. If it is at a low level, the processor is ready to receive ASCII commands and characters, but if it is at a high level, commands are being processed.

RS232Tx (pin 17)

This is the RS232 data transmit output. The signal level is compatible with most interface ICs (output is normally high), and there is sufficient current drive to allow interfacing using only a PNP transistor, if desired.

RS232Rx (pin 18)

This is the RS232 receive data input. The signal level is compatible with most interface ICs (when at idle, the level is normally high), but can be used with other interfaces as well, since the input has Schmitt trigger waveshaping.

V_{DD} (pin 20)

This pin is the positive supply pin, and should always be the most positive point in the circuit. Internal circuitry connected to this pin is used to provide power on reset of the microprocessor, so an external reset signal is not required. Refer to the Electrical Characteristics section for further information.

ISO K (pin 21) and ISO L (pin 22)

These are the active high output signals which are used to drive the ISO 9141 and ISO 14230 buses to an active (dominant) level. Many new vehicles do not require the L Line – if yours does not, you can simply leave pin 22 open-circuited.

CAN Tx (pin 23) and CAN Rx (pin 24)

These are the two CAN interface signals that must be connected to a CAN transeiver IC (see the Example Applications section for more information). If unused, pin 24 should be connected to a logic high (V_{DD}) level.

RS232 Rx LED (pin 25), RS232 Tx LED (pin 26),
OBD Rx LED (pin 27) and OBD Tx LED (pin 28)

These four output pins are normally high, and are driven to low levels when the ELM327 is transmitting or receiving data. These outputs are suitable for directly driving most LEDs through current limiting resistors, or interfacing to other logic circuits. If unused, these pins may be left open-circuited.

Note that pin 28 can also be used to turn off all of the Programmable Parameters - see page 45 for details.

Ordering Information

These integrated circuits are 28 pin devices, available in either a 300 mil wide plastic (‘skinny’) DIP format or in a 300 mil SOIC surface mount type of package. To order, add the appropriate suffix to the part number:

300 mil 28 pin Plastic DIP.....ELM327P

300 mil 28 pin SOIC.....ELM327SM

Absolute Maximum Ratings

Storage Temperature..... -65°C to +150°C

Ambient Temperature with
Power Applied.....-40°C to +85°C

Voltage on V_{DD} with respect to V_{SS}..... -0.3V to +7.5V

Voltage on any other pin with
respect to V_{SS}..... -0.3V to (V_{DD} + 0.3V)

Note:

These values are given as a design guideline only. The ability to operate to these levels is neither inferred nor recommended, and stresses beyond those listed here will likely damage the device.

Electrical Characteristics

All values are for operation at 25°C and a 5V supply, unless otherwise noted. For further information, refer to note 1 below.

Characteristic	Minimum	Typical	Maximum	Units	Conditions
Supply voltage, V_{DD}	4.5	5.0	5.5	V	
V_{DD} rate of rise	0.05			V/ms	see note 2
Average supply current, I_{DD}		9		mA	see note 3
Input threshold voltage	1.0		1.3	V	all except Schmitt inputs
Schmitt trigger input thresholds rising		2.9	4.0	V	see note 4
Schmitt trigger input thresholds falling	1.0	1.5		V	see note 4
Output low voltage		0.3		V	current (sink) = 10 mA
Output high voltage		4.6		V	current (source) = 10 mA
Brown-out reset voltage	4.11	4.33	4.55	V	
A/D conversion time		7		msec	see note 5

Notes:

1. This integrated circuit is produced with one of Microchip Technology Inc.'s PIC18F2x8x family of devices as the core embedded microcontroller. For further device specifications, and possibly clarification of those given, please refer to the appropriate Microchip documentation (available at http://www.microchip.com/).
2. This spec must be met in order to ensure that a correct power on reset occurs. It is quite easily achieved using most common types of supplies, but may be violated if one uses a slowly varying supply voltage, as may be obtained through direct connection to solar cells or some charge pump circuits.
3. Device only. Does not include any load currents.
4. Pins 1, 11, 12, 13, 15 and 18 (only) have internal Schmitt trigger waveshaping circuitry. All other inputs use standard CMOS (TTL compatible) circuitry.
5. The typical width of the Busy output pulse while the ELM327 interprets the command, measures the voltage, scales it and then transmits the result of a mid-range measurement, with the RS232 rate at 38400 baud.

Overview

The following describes how to use the ELM327 to obtain information from your vehicle.

We begin by discussing just how to “talk” to the IC using a PC, then explain how to change options using ‘AT’ commands, and finally we show how to use the ELM327 to obtain trouble codes (and reset them). For the more advanced experimenters, there are also sections on how to use some of the programmable features of this product as well.

Using the ELM327 is not as daunting as it first seems. Many users will never need to issue an ‘AT’ command, adjust timeouts, or change the headers. For most, all that is required is a PC or a PDA with a terminal program (such as HyperTerminal or ZTerm), and knowledge of one or two OBD commands, which we will provide in the following sections…

Communicating with the ELM327

The ELM327 expects to communicate with a PC through an RS232 serial connection. Although modern computers do not usually provide a serial connection such as this, there are several ways in which a ‘virtual serial port’ can be created. The most common devices are USB to RS232 adapters, but there are several others such as PC cards, ethernet devices, or Bluetooth to serial adapters.

No matter how you physically connect to the ELM327, you will need a way to send and receive data. The simplest method is to use one of the many ‘terminal’ programs that are available (HyperTerminal, ZTerm, etc.), to allow typing the characters directly from your keyboard.

To use a terminal program, you will need to make several settings. First, ensure that your software is set to use the proper ‘COM’ port, and that you have chosen the proper data rate - this will be either 9600 baud (if pin 6=0V at power up), or 38400 baud (if PP 0C has not been changed). If you select the wrong “COM” port, you will not be able to send or receive any data. If you select the wrong data rate, the information that you send and receive will be all garbled, and unreadable by you or the ELM327. Don’t forget to also set your connection for 8 data bits, no parity bits, and 1 stop bit, and to set it for the proper “line end” mode. All of the responses from the ELM327 are terminated with a single carriage return character and, optionally, a linefeed character (depending on your settings).

Properly connected and powered, the ELM327 will energize the four LED outputs in sequence (as a lamp test) and will then send the message:

```
ELM327 v1.3a

>
```

In addition to identifying the version of this IC, receiving this string is a good way to confirm that the computer connections and terminal software settings are correct (however, at this point no communications have taken place with the vehicle, so the state of that connection is still unknown).

The ‘>’ character that is shown on the second line is the ELM327’s prompt character. It indicates that the device is in the idle state, ready to receive characters on the RS232 port. If you did not see the identification string, you might try resetting the IC again with the AT Z (reset) command. Simply type the letters A T and Z (spaces are optional), then press the return key:

```
>AT Z
```

That should cause the leds to flash again, and the identification string to be printed. If you see strange looking characters, then check your baud rate - you have likely set it incorrectly.

Characters sent from the computer can either be intended for the ELM327’s internal use, or for reformatting and passing on to the vehicle. The ELM327 can quickly determine where the received characters are to be directed by monitoring the contents of the message. Commands that are intended for the ELM327’s internal use will begin with the characters ‘AT’, while OBD commands for the vehicle are only allowed to contain the ASCII codes for hexadecimal digits (0 to 9 and A to F).

Whether it is an ‘AT’ type internal command or a hex string for the OBD bus, all messages to the ELM327 must be terminated with a carriage return character (hex ‘0D’) before it will be acted upon. The one exception is when an incomplete string is sent and no carriage return appears. In this case, an internal timer will automatically abort the incomplete message after about 20 seconds, and the ELM327 will print a single question mark (‘?’) to show that the input was not understood (and was not acted upon).

Messages that are not understood by the ELM327 (syntax errors) will always be signalled by a single

Communicating with the ELM327 (continued)

question mark. These include incomplete messages, incorrect AT commands, or invalid hexadecimal digit strings, but are not an indication of whether or not the message was understood by the vehicle. One must keep in mind that the ELM327 is a protocol interpreter that makes no attempt to assess the OBD messages for validity – it only ensures that hexadecimal digits were received, combined into bytes, then sent out the OBD port, and it does not know if a message sent to the vehicle was in error.

While processing OBD commands, the ELM327 will continually monitor for either an active RTS input, or an RS232 character received. Either one will interrupt the IC, quickly returning control to the user, while possibly aborting any initiation, etc. that was in progress. After generating a signal to interrupt the ELM327, software should always wait for either the prompt character ('>' or hex 3E), or a low level on the Busy output before beginning to send the next command.

Finally, it should be noted that the ELM327 is not case-sensitive, so the commands 'ATZ', 'atz', and 'AtZ' are all exactly the same to the ELM327. All commands may be entered as you prefer, as no one method is faster or better. The ELM327 also ignores space characters and all control characters (tab, etc.) in the input, so they can be inserted anywhere in the input if that improves readability.

One other feature of the ELM327 is the ability to repeat any command (AT or OBD) when only a single carriage return character is received. If you have sent a command (for example, 01 0C to obtain the rpm), you do not have to resend the entire command in order to resend it to the vehicle - simply send a carriage return character, and the ELM327 will repeat the command for you. The memory buffer only remembers the one last command - there is no provision in the current ELM327 to provide storage for any more.

Please Note:

There is a very small chance that NULL characters (byte value 00) may occasionally be inserted into the RS232 data that is transmitted by the ELM327.

Microchip Technology has reported that some ICs which use the same EUSART as in the ELM327 may, under very specific (and rare) conditions, insert an extra byte (always of value 00) into the transmitted data. If you are using a terminal program to view the data, you should select the 'hide control characters' option if it is available, and if you are writing software for the ELM327, then monitor incoming bytes, and ignore any that are of value 00 (ie. remove NULLs).

AT Commands

Several parameters within the ELM327 can be adjusted in order to modify its behaviour. These do not normally have to be changed before attempting to talk to the vehicle, but occasionally the user may wish to customize these settings – for example by turning the character echo off, adjusting a timeout value, or changing the header bytes. In order to do this, internal 'AT' commands must be used.

Those familiar with PC modems will immediately recognize AT commands as a standard way in which modems are internally configured. The ELM327 uses essentially the same method, always watching 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 an internal configuration or 'AT' command, and will be executed upon receipt of a terminating carriage return character. If the command is just a setting change, the ELM327 will reply with the characters 'OK', to say that it was successfully completed.

Some of the following commands allow passing numbers as arguments in order to set the internal values. These will always be hexadecimal numbers which must generally be provided in pairs. The hexadecimal conversion chart in the OBD Commands section (page 23) may prove useful if you wish to interpret the values. Also, one should be aware that for the on/off types of commands, the second character is the number 1 or the number 0, the universal terms for on and off.

The remainder of this page, and the page following provide a summary of all of the commands that the current version of the ELM327 recognizes. A more complete description of each command is provided on the pages following.

AT Command Summary

General Commands

<CR>	repeat the last command
BRD hh	try Baud Rate Divisor hh
BRT hh	set Baud Rate Timeout
D	set all to Defaults
E0, E1	Echo Off, or On*
FE	Forget Events
I	print the version ID
L0, L1	Linefeeds Off, or On
M0, M1	Memory Off, or On
WS	Warm Start (quick software reset)
Z	reset all
@1	display the device description
@2	display the device identifier
@3 ccccccccccc	store the device identifier

* = default setting

Programmable Parameter Commands

PP xx OFF	disable Prog Parameter xx
PP FF OFF	all Prog Parameters Off
PP xx ON	enable Prog Parameter xx
PP FF ON	all Prog Parameters On
PP xx SV yy	for PP xx, Set the Value to yy
PPS	print a PP Summary

Voltage Reading Commands

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

the Commands continue on the following page…

AT Command Summary (continued)

OBD Commands

AL	Allow Long (>7 byte) messages
AR	Automatically Receive
AT0, 1, 2	Adaptive Timing Off, Auto1*, Auto2
BD	perform a Buffer Dump
BI	Bypass the Initialization sequence
DP	Describe the current Protocol
DPN	Describe the Protocol by Number
H0, H1	Headers Off*, or On
MA	Monitor All
MR hh	Monitor for Receiver = hh
MT hh	Monitor for Transmitter = hh
NL	Normal Length messages*
PC	Protocol Close
R0, R1	Responses Off, or On*
RA hh	set the Receive Address to hh
S0, S1	printing of Spaces Off, or On*
SH xyz	Set Header to xyz
SH xxyyzz	Set Header to xxyyzz
SP h	Set Protocol to h and save it
SP Ah	Set Protocol to Auto, h and save it
SR hh	Set the Receive address to hh
ST hh	Set Timeout to hh x 4 msec
TP h	Try Protocol h
TP Ah	Try Protocol h with Auto search

J1850 Specific Commands (protocols 1 and 2)

IFR0, 1, 2	IFRs Off, Auto*, or On
IFR H, S	IFR value from Header* or Source

ISO Specific Commands (protocols 3 to 5)

IB 10	Set the ISO Baud rate to 10400*
IB 96	Set the ISO Baud rate to 9600
IIA hh	Set the ISO (slow) Init Address to hh
KW	display the Key Words
KW0, KW1	Key Word checking Off, or On*
SW hh	Set Wakeup interval to hh x 20 msec
WM [1 - 6 bytes]	Set the Wakeup Message

CAN Specific Commands (protocols 6 to C)

CAF0, CAF1	Automatic Formatting Off, or On*
CF hhh	set the ID Filter to hhh
CF hhhhhhhh	set the ID Filter to hhhhhhhh
CFC0, CFC1	Flow Controls Off, or On*
CM hhh	set the ID Mask to hhh
CM hhhhhhhh	set the ID Mask to hhhhhhhh
CP hh	set CAN Priority to hh (29 bit)
CRA hhh	set CAN Receive Address to hhh
CRA hhhhhhhh	set the Rx Address to hhhhhhhh
CS	show the CAN Status counts
D0, D1	display of the DLC Off*, or On
FC SM h	Flow Control, Set the Mode to h
FC SH hhh	FC, Set the Header to hhh
FC SH hhhhhhhh	FC, Set the Header to hhhhhhhh
FC SD [1 - 5 bytes]	FC, Set Data to [...]
RTR	send an RTR message
V0, V1	use of Variable DLC Off*, or On

J1939 CAN Specific Commands (protocols A to C)

DM1	Monitor for DM1 messages
JE	use J1939 Elm data format*
JS	use J1939 SAE data format
MP hhhh	Monitor for PGN 0hhhh
MP hhhhhh	Monitor for PGN hhhhhh

* = default setting

AT Command Descriptions

The following describes each AT Command that the current version of the ELM327 supports:

AL [Allow Long messages]

The standard OBDII protocols restrict the number of data bytes in a message to seven, which the ELM327 normally does as well (for both send and receive). If AL is selected, the ELM327 will allow long sends (eight data bytes) and long receives (unlimited in number). The default is AL off (and NL selected).

AR [Automatically set the Receive address]

Responses from the vehicle will be acknowledged and displayed by the ELM327, if its internally stored receive address matches the address that the message is being sent to. With the auto receive mode in effect, the value used for the receive address will be chosen based on the current header bytes, and will automatically be updated whenever the header bytes are changed.

The value that is used for the receive address is determined based on such things as the contents of the first header byte, and whether the message uses physical addressing, functional addressing, or if the user has set a value with the SR or RA commands.

Auto Receive is turned on by default, and is not used by the J1939 formatting.

AT0, **AT1** and **AT2** [Adaptive Timing control]

When receiving responses from a vehicle, the ELM327 has traditionally waited the time set by the AT ST hh setting for a response. To ensure that the IC would work with a wide variety of vehicles, the default value was set to a conservative (slow) value. Although it was adjustable, many people did not have the equipment or experience to determine a better value.

The Adaptive Timing feature will automatically set the timeout value for you, to a value that is based on the actual response times that your vehicle is responding in. As conditions such as bus loading, etc. change, the algorithm learns from them, and makes appropriate adjustments. Note that it always uses your AT ST hh setting as the maximum setting, and will never choose one which is longer.

There are three adaptive timing settings that are available for use. By default, Adaptive Timing option 1 (AT1) is enabled, and is the recommended setting. AT0 is used to disable Adaptive timing (usually used when experimenting), while AT2 is a more agressive version of AT1 (the effect is more noticeable for very slow connections – you may not see much difference with faster OBD systems). The J1939 protocol does not support Adaptive Timing – responses for J1939 use fixed timeouts as set in the standard.

BD [perform an OBD Buffer Dump]

All messages sent and received by the ELM327 are stored temporarily in a set of twelve memory storage locations called the OBD Buffer. Occasionally, it may be of use to view the contents of this buffer, perhaps to see why an initiation failed, to see the header bytes in the last message, or just to learn more of the structure of OBD messages. You can ask at any time for the contents of this buffer to be 'dumped' (printed) – when you do, the ELM327 sends a length byte (representing the length of the message in the buffer) followed by the contents of all twelve OBD buffer locations.

The length byte represents the actual number of bytes received, whether they fit into the OBD buffer or not. This may be useful when viewing long data streams (with AT AL), as the number accurately represents the number of bytes received, mod 256. Note that only the first twelve bytes received are stored in the buffer.

BI [Bypass the Initialization sequence]

This command should be used with caution. It allows an OBD protocol to be made active without requiring any sort of initiation or handshaking to occur. The initiation process is normally used to validate the protocol, and without it, results may be difficult to predict. It should not be used for routine OBD use, and has only been provided to allow the construction of ECU simulators and training demonstrators.

BRD hh [try Baud Rate Divisor hh]

This command is used to change the RS232 baud rate divisor to the hex value provided by hh. The actual baud rate (in kbps) will be 4000 divided by this divisor. For example, a setting of 115.2kbps would require a divisor of 4000/115.2 or 35. In hexadecimal notation,

AT Command Descriptions (continued)

35 is written as 23, so the actual command that needs to be sent would be AT BRD 23.

Since the ELM327 may be able to operate at much higher rates than some interfaces can support, the BRD command allows requested rates to be tested before they are committed to (with automatic fall-back to the previous baud rate if there are problems). In use, the command is sent as a request to change the baud rate, and the ELM327 responds with the familiar "OK". After that, an internal timer begins waiting, to ensure that the controlling computer has sufficient time to change their baud rate to the same rate. The ELM327 then sends the poweron message at the new baud rate, and begins waiting while the controlling computer assesses what has been received. If the AT I message was received without errors, the controlling computer sends a carriage return character, to test the transmit circuitry. If received correctly by the ELM327, the new rate is retained. If the controlling computer should see errors, however, it simply sends no response, the ELM327 waits the time set by AT BRT, and when no response is detected, reverts back to the former baud rate. A more detailed discussion of this entire process is provided in the 'Using Higher RS232 Baud Rates' section, on pages 48 and 49.

Any new baud rate that is set in this manner is retained across calls to set defaults (AT D), and for warm starts (AT WS), but will not survive a hardware reset (a power off/on or a call to AT Z). If you are in the habit of calling AT Z in your code, we advise using AT WS instead.

BRT hh [set Baud Rate Timeout to hh]

This command allows the timeout used for the Baud Rate handshake (ie. AT BRD) to be varied. The time delay is given by hh x 5.0 msec, where hh is a hexadecimal value. The default value for this setting is 0F, providing 75msec. Note that a value of 00 does not result in 0 msec - it provides the maximum time of 256 x 5.0 msec, or 1.28 seconds.

CAF0 and **CAF1** [CAN Auto Formatting off or on]

These commands determine whether the ELM327 assists you with the formatting of the CAN data that is sent and received. With CAN Automatic Formatting enabled (CAF1), the IC will automatically generate formatting (PCI) bytes for you when sending, and will remove them when receiving. This means that you can continue to issue OBD requests (01 00, etc.) as usual, without regard to the extra bytes that CAN diagnostics systems require. Also, with formatting on, any extra (unused) data bytes that are received in the frame will be removed, and any messages with invalid PCI bytes will be ignored. When monitoring, however, messages with invalid PCI bytes will be shown with a '<DATA ERROR' statement to show that the data is not valid for the chosen protocol.

Occasionally, long (multi-frame) responses are returned by the vehicle. To help you analyze these, the Auto Formatting mode will extract the total data length and print it on one line. Following this will be each segment of the message, with the segment number (a single hexadecimal digit) shown at the beginning of the line with a colon (':') as a separator.

You may also see the characters 'FC: ' at the beginning of a line (if you are experimenting). This represents a Flow Control message that is sent as part of the multi-line message signalling. Flow Control messages are automatically generated by the ELM327 in response to a "First Frame" reply, as long as the CFC setting is on (it does not matter whether you have selected the CAF1 or the CAF0 modes).

Another type of message – the RTR (or 'Remote Transfer Request') – will be automatically hidden for you when in the CAF1 mode, since they contain no data. When auto formatting is off (CAF0), you will see the characters 'RTR' printed when a remote transfer request frame has been received.

Turning the CAN Automatic Formatting off (CAF0), will cause the ELM327 to print all of the received data bytes. No bytes will be hidden from you, and none will be inserted for you. Similarly, when sending a data request with formatting off, you must provide all of the required data bytes exactly as they are to be sent – the ELM327 will not perform any formatting for you other than to add some trailing 'padding' bytes to ensure that the required eight data bytes are sent. This allows operation in systems that do not use PCI bytes as ISO 15765-4 does.

Note that turning the display of headers on (with AT H1) will override some of the CAF1 formatting of the received data frames, so that the received bytes will appear much like in the CAF0 mode (ie. as received). It is only the printing of the received data that will be affected when both CAF1 and H1 modes are enabled, though; when sending data, the PCI byte will still be created for you and padding bytes will still be added. Auto Formatting on (CAF1) is the default setting for the ELM327.

AT Command Descriptions (continued)

CF hhh [set the CAN ID Filter to hhh]

The CAN Filter works in conjunction with the CAN Mask to determine what information is to be accepted by the receiver. As each message is received, 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 ELM327, otherwise it will be discarded. This three nibble version of the CAN Filter command makes it a little easier to set filters with 11 bit ID CAN systems. Only the rightmost 11 bits of the provided nibbles are used, and the most significant bit is ignored. The data is actually stored as four bytes internally however, with this command adding leading zeros for the other bytes. See the CM command(s) for more details.

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

This command allows all four bytes (actually 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. Note that this command may be used to enter 11 bit ID filters as well, since they are stored in the same locations internally (entering AT CF 00 00 0h hh is exactly the same as entering the shorter AT CF hhh command).

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 ELM327 automatically sends these, without any intervention by the user. If experimenting with a non-OBD system, it may be desirable to turn this automatic response off, and the AT CFC0 command has been provided for that purpose. The default setting is CFC1 - Flow Controls on.

Note that during monitoring (AT MA, MR, or MT), there are never any Flow Controls sent no matter what the CFC option is set to.

CM hhh [set the CAN ID Mask to hhh]

There can be a great many messages being transmitted in a CAN system at any one time. In order to limit what the ELM327 views, there needs to be a system of filtering out the relevant ones from all the others. This is accomplished by the filter, which works in conjunction with the mask. A mask is a group of bits that show the ELM327 which bits in the filter are relevant, and which ones can be ignored. A 'must match' condition is signaled by setting a mask bit to '1', while a 'don't care' is signaled by setting a bit to '0'. This three digit variation of the CM command is used to provide mask values for 11 bit ID systems (the most significant bit is always ignored).

Note that a common storage location is used internally for the 29 bit and 11 bit masks, so an 11 bit mask could conceivably be assigned with the next command (CM hh hh hh hh), should you wish to do the extra typing. The values are right justified, so you would need to provide five leading zeros followed by the three mask bytes.

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

This command is used to assign mask values for 29 bit ID systems. See the discussion under the CM hhh command – it is essentially identical, except for the length. Note that the three most significant bits that you provide in the first digit will be ignored.

CP hh [set CAN Priority bits to hh]

This command is used to set the five most significant bits of a 29 bit CAN ID word (the other 24 bits are set with the AT SH command). Many systems use several of these bits to assign a priority value to messages, which is how the command was named. Any bits provided in excess of the five required will be ignored, and not stored by the ELM327 (it only uses the five least significant bits of this byte). The default value for these priority bits is hex 18.

CRA hhh [set the CAN Rx Addr to hhh]

Setting the CAN masks and filters can be difficult at times, so if you only want to receive information from one address (ie. one CAN ID), then this command may be very welcome. For example, if you only want to see information from 7E8, simply send AT CRA 7E8, and the ELM327 will make the necessary adjustments to both the mask and the filter for you.

Note that this command restricts viewing to only the one ID - to allow reception of a range of IDs, you must set the mask and filter independently (described in detail on page 36). To reverse the changes made by the CRA command, simply send the Auto Receive command (AT AR).

AT Command Descriptions (continued)

CRA hhhhhhhh [set the CAN Rx Addr to hhhhhhhh]

This command is identical to the previous one, except that it used to set 29 bit CAN IDs, instead of 11. The Auto Receive command (AT AR) is also used to reverse the changes made by this command.

CS [show the CAN Status counts]

The CAN protocol requires that statistics be kept regarding the number of transmit and receive errors detected. If there should be a significant number of errors (due to a hardware or software problem), the device will go off-line in order to not affect other data on the bus. The AT CS command lets you see both the transmitter (Tx) and the receiver (Rx) error counts, in hexadecimal. If the transmitter should be off (count >FF), you will see 'OFF' rather than a specific count.

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

The voltage reading that the ELM327 shows for an AT RV request can be calibrated with this command. The argument ('dddd') must always be provided as 4 digits, with no decimal point (it assumes that the decimal place is between the second and the third digits).

To use this feature, simply use an accurate meter to read the actual input voltage, then use the CV command to change the internal calibration (scaling) factor. For example, if the ELM327 shows the voltage as 12.2V while you measure 11.99 volts, then send AT CV 1199 and the ELM327 will recalibrate itself for that voltage (it will actually read 12.0V due to digit roundoff). See page 22 for some more information on how to read voltages and perform the calibration.

D [set all to Defaults]

This command is used to set the options to their default (or factory) settings, as when power is first applied. The last stored protocol will be retrieved from memory, and will become the current setting (possibly closing other protocols that are active). 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.

D0 and **D1** [display DLC off (0) or on (1)]

Standard CAN (ISO 15765-4) OBD requires that all messages have 8 data bytes, so displaying the number of data bytes (the DLC) is not normally very useful. When experimenting with other protocols, however, it may be useful to be able to see what the data lengths are. The D0 and D1 commands control the display of the DLC digit (the headers must also be on, in order to see this digit). When displayed, the single DLC digit will appear between the ID (header) bytes and the data bytes. The default setting is determined by PP 29.

DM1 [monitor for DM1s]

The SAE J1939 Protocol broadcasts trouble codes periodically as they are detected, using Diagnostic Mode 1 (DM1) messages. This command sets the ELM327 to continually monitor for this type of message for you, following multi-segment transport protocols as required. Note that a combination of masks and filters could be set to provide a similar output, but they would not allow multiline messages to be detected. The DM1 command adds the extra logic that is needed for multiline messages.

This command is only available when a CAN Protocol (A, B, or C) has been selected for J1939 formatting. It returns an error if attempted under any other conditions.

DP [Describe the current Protocol]

The ELM327 is capable of automatically determining the appropriate OBD protocol to use for each vehicle that it is connected to. When the IC connects to a vehicle, however, it returns only the data requested, and does not report the protocol found. The DP command is used to display the current protocol that the ELM327 is selected for (even if it is not connected). If the automatic option is also selected, the protocol will show the word "AUTO" before it, followed by the type. Note that the actual protocol names are displayed, not the numbers used by the protocol setting commands.

DPN [Describe the Protocol by Number]

This command is similar to the DP 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 set protocol and test protocol commands.

AT Command Descriptions (continued)

E0 and **E1** [Echo off (0) or on (1)]

These commands control whether or not the characters received on the RS232 port are echoed (retransmitted) back to the host computer. Character echo can be used to confirm that the characters sent to the ELM327 were received correctly. The default is E1 (or echo on).

FC SM h [Flow Control Set Mode to h]

This command sets how the ELM327 responds to First Frame messages when automatic Flow Control responses are enabled. The single digit provided can either be '0' (the default) for fully automatic responses, '1' for completely user defined responses, or '2' for user defined data bytes in the response. More complete details and examples can be found in the Altering Flow Control Messages section (page 40).

FC SH hhh [Flow Control Set Header to…]

The header (or more properly 'CAN ID') bytes used for CAN Flow Control response messages can be set using this command. Only the right-most 11 bits of those provided will be used - the most significant bit is always removed. This command only affects Flow Control mode 1.

FC SH hhhhhhhh [Flow Control Set Header to…]

This command is used to set the header (or 'CAN ID') bits for Flow Control responses with 29 bit CAN ID systems. Since the 8 nibbles define 32 bits, only the right-most 29 bits of those provided will be used - the most significant three bits are always removed. This command only affects Flow Control mode 1.

FC SD [1-5 bytes] [Flow Control Set Data to…]

The data bytes that are sent in a CAN Flow Control message can be set with this command. The current version of the software allows one to five data bytes to be defined, with the remainder of the data bytes in the message being automatically set to the default CAN filler byte. Data provided with this command is only used when Flow Control modes 1 or 2 have been enabled.

FE [Forget Events]

There are certain events which may change how the ELM327 responds from that point on. One of these is the occurance of an ERR94 condition, that blocks subsequent searching through CAN protocols if PP 2A bit 5 is '1'. Normally, an event such as this will affect all searches until the next power off and on, but it can be 'forgotten' using software, with the AT FE command. Similarly, an 'LV RESET' event will prevent searches through CAN protocols if PP 2A bit 4 is '1', and it may also be forgotten with the FE command. FE is a new command with v1.3a of the IC.

H0 and **H1** [Headers off (0) or on (1)]

These commands control whether or not the additional (header) bytes of information are shown in the responses from the vehicle. These are normally not shown by the ELM327, but can be turned on by issuing the AT H1 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, and possibly the CAN data length code (DLC) if it has been enabled with PP 29. The current version of this IC does not display the CAN CRC code, nor the special J1850 IFR bytes (which some protocols use to acknowledge receipt of a message).

I [Identify yourself]

Issuing this command causes the chip to identify itself, by printing the startup product ID string (currently "ELM327 v1.3a"). Software can use this to determine exactly which integrated circuit it is talking to, without having to reset the IC.

IB 10 [set the ISO Baud rate to 10400]

This command restores the ISO 9141-2 and ISO 14230-4 baud rates to the default value of 10400.

IB 96 [set the ISO Baud rate to 9600]

Several users have requested this command. It is used to change the baud rate used for the ISO 9141-2 and ISO 14230-4 protocols (numbers 3, 4, and 5) to 9600 baud, while relaxing some of the requirements for the initiation byte transfers. It may be useful for experimenting with some vehicles. Normal 10,400 baud operation can be restored at any time by issuing an IB 10 command.

AT Command Descriptions (continued)

IFR0, IFR1, and **IFR2** [IFR control]

The SAE J1850 protocol allows for an In-Frame Response (IFR) byte to be sent after each message, usually to acknowledge the correct receipt of that message. The ELM327 automatically generates and sends this byte for you by default, but you can override this behaviour with this command.

The AT IFR0 command will disable the sending of all IFRs, no matter what the header bytes require. AT IFR2 is the opposite - it will force an IFR byte to always be sent, no matter what header bytes indicate. The AT IFR1 command restores the response to provide the automatic sending of IFRs, as determined by the 'K' bit of the header byte. IFR1 is the default setting of the ELM327.

IFR H and **IFR S** [IFR from Header or Source]

The value sent in the J1850 In-Frame Response (IFR) byte is normally the same as the value sent as the Source (or Tester) Address byte that was in the header of the request. There may be occasions when it is desireable to use some other value, however, and this set of commands allows for this.

If you send AT IFR S, the ELM327 will use the value defined as the Source Address (usually F1, but it can be changed with PP 06), even if another value was sent in the Header bytes. This is not what is normally required, and caution should be used when using AT IFR S. AT IFR H restores the sending of the IFR bytes to those provided in the Header, and is the default setting.

IIA hh [set the ISO Init Address to hh]

The ISO 9141-2 and ISO 14230-4 standards state that when beginning a session with an ECU, the initiation sequence is to be directed to a specific address ($33). If you wish to experiment by directing the slow five baud sequence to another address, it is done with this command. For example, if you prefer that the initiation be performed with the ECU at address $7A, then simply send:

```
>AT IIA 7A
```

and the ELM327 will use that address when called to do so (protocols 3 or 4). The full eight bit value is used exactly as provided – no changes are made to it (ie no adding of parity bits, etc.)

Note that setting this value does not affect any address values used in the header bytes. The ISO init address is restored to $33 whenever the defaults, or the ELM327, are reset.

JE [enables the J1939 ELM data format]

The J1939 standard requires that PGN requests be sent with the byte order reversed from the standard 'left-to-right' order, which many of us would expect. For example, to send a request for the engine temperature (PGN 00FEEE), the data bytes are actually sent in the reverse order (ie EE FE 00), and the ELM327 would normally expect to receive the data in that order for passing on to the vehicle.

When experimenting, this constant need for byte reversals can be quite confusing, so we have defined an ELM format that reverses the bytes for you. When the J1939 ELM (JE) format is enabled, and you have a J1939 protocol selected, and you provide three data bytes to the ELM327, it will reverse the order for you before sending them to the ECU. To request the engine temperature PGN, you would send 00 FE EE (and not EE FE 00). The 'JE' type of automatic formatting is enabled by default.

JS [enables the J1939 SAE data format]

The AT JS command disables the automatic byte reordering that the JE command performs for you. If you wish to send data bytes to the J1939 vehicle without any manipulation of the byte order, then select JS formatting.

Using the above example for engine temperature (PGN 00FEEE) with the data format set to JS, you must send the bytes to the ELM327 as EE FE 00 (this is also known as little-endian byte ordering).

The JS type of data formatting is off by default, but was the only type of data formatting provided by the ELM327 v1.2. If you are switching from version 1.2 of the IC, take note of this difference.

KW [display the Key Words]

When the ISO 9141-2 and ISO 14230-4 protocols are initialized, two special bytes (key words) are passed to the ELM327 (the values are used internally to determine whether a particular protocol variation can be supported by the ELM327). If you wish to see what the value of these bytes were, simply send the AT KW command.

AT Command Descriptions (continued)

KW0 and **KW1** [Key Word checks off (0) or on (1)]

The ELM327 looks for specific bytes (called Key Words) to be sent to it during the ISO 9141-2 and ISO14230-4 initiation sequences. If those bytes are not found, the initiation is said to have failed (you might see “UNABLE TO CONNECT” or perhaps “BUS INIT: ...ERROR”). This may be because you are trying to connect to a non-OBD compliant ECU, or perhaps to an older one.

If you wish to experiment, but do not want the ELM327 to check the values contained in the key words, you can turn the checking off with:

```
>AT KW0
```

after which the IC will look for Key Word bytes in the response, but will not look at the actual values of the bytes. This may allow a connection in an otherwise ‘impossible’ situation. Normal behaviour can be returned with AT KW1, which is the default setting.

Caution should be used with this command, as you are bypassing the checks that are normally performed on the keyword bytes. The ELM327 sends an acknowledgement to the ECU for these bytes, but that is without considering what the bytes actually are. You could be incorrectly activating an ISO 9141, or KWP 2000 protocol, so should be very careful.

L0 and **L1** [Linefeeds off (0) or on (1)]

This option controls the sending of linefeed characters after each carriage return character. For AT L1, linefeeds will be generated after every carriage return character, and for AT L0, they will be off. Users will generally wish to have this option on if using a terminal program, but off if using a custom computer interface (as the extra characters transmitted will only serve to slow the communications down). The default setting is determined by the voltage at pin 7 during power on (or reset). If the level is high, then linefeeds on will be the default; otherwise it will be linefeeds off.

M0 and **M1** [Memory off (0) or on (1)]

The ELM327 has internal ‘non-volatile’ memory that is capable of remembering the last protocol used, even after the power is turned off. This can be convenient if the IC is often used for one particular protocol, as that will be the first one attempted when next powered on. To enable this memory function, it is necessary to either use an AT command to select the M1 option, or to have chosen “memory on” as the default power on mode (by connecting pin 5 of the ELM327 to a high logic level).

When the memory function is enabled, each time that the ELM327 finds a valid OBD protocol, that protocol will be memorized (stored) and will become the new default. If the memory function is not enabled, protocols found during a session will not be memorized, and the ELM327 will always start at power up using the same (last saved) protocol.

If the ELM327 is to be used in an environment where the protocol is constantly changing, it would likely be best to turn the memory function off, and issue an AT SP 0 command once. The SP 0 command tells the ELM327 to start in an 'Automatic' protocol search mode, which is the most useful for an unknown environment. ICs come from the factory set to this mode. If, however, you have only one vehicle that you regularly connect to, storing that vehicle’s protocol as the default would make the most sense.

As mentioned, the default setting for the memory function is determined by the voltage level at pin 5 at power up (or system reset). If it is connected to a high level (V_{DD}), then the memory function will be on by default. If pin 5 is connected to a low level, the memory saving will be off by default.

MA [Monitor All messages]

This command places the ELM327 into a bus monitoring mode, in which it continually monitors for (and displays) all messages that it sees on the OBD bus. It is a quiet monitor, not sending In Frame Responses for J1850 systems, Acknowledges for CAN systems, or Wakeup (‘keep-alive’) messages for the ISO 9141 and ISO 14230 protocols. Monitoring will continue until it is stopped by activity on the RS232 input, or the RTS pin.

To stop the monitoring, simply send any single character to the ELM327, then wait for it to respond with a prompt character (‘>’), or a low level output on the Busy pin. (Setting the RTS input to a low level will interrupt the device as well.) Waiting for the prompt is necessary as the response time varies depending on what the IC was doing when it was interrupted. If for instance it is in the middle of printing a line, it will first complete that line then return to the command state, issuing the prompt character. If it were simply waiting for input, it would return immediately. Note that the character which stops the monitoring will always be discarded, and will not affect subsequent commands.

Beginning with v1.3 of this IC, all messages will be

AT Command Descriptions (continued)

printed as found, even if the CAN auto formatting is on (CAF1). The previous version of this IC (v1.2) did not display some illegal CAN messages if the automatic formatting was on, but now all messages received are displayed, and if the data format does not appear to be correct, then "<DATA ERROR" will be shown beside the data.

If this command is used with CAN protocols, and if the CAN filter and/or mask were previously set (with CF, CM or CRA), then the MA command will be affected by the settings. For example, if the receive address had been set previously with CRA 4B0, then the AT MA command would only be able to 'see' messages with an ID of 4B0. This is often not what is desired - you may want to reset the masks and filters (with AT AR) first.

All of the monitoring commands (MA, MR and MT) operate by closing the current protocol (an AT PC is executed internally), then configuring the IC for silent monitoring of the data (no wakeup messages, IFRs or CAN acknowledges are sent by the ELM327). When the next OBD command is to be transmitted, the protocol will again be initialized, and you may see messages stating this. "SEARCHING..." may also be seen, depending on what changes were made while monitoring.

MP hhhh [Monitor for PGN hhhh]

The AT MA, MR and MT commands are quite useful for when you wish to monitor for a specific byte in the header of a typical OBD message. For the SAE J1939 Protocol, however, it is often desireable to monitor for the multi-byte Parameter Group Numbers (or PGNs), which can appear in either the header, or the data bytes. The MP command is a special J1939 only command that is used to look for responses to a particular PGN request, and follow any multi-segment occurances of them.

Note that this MP command provides no means to set the first two digits of the requested PGN, and they are always assumed to be 00. For example, the DM2 PGN has an assigned value of 00FECB (see SAE J1939-73). To monitor for DM2 messages, you would issue AT MP FECB, eliminating the 00, since the MP hhhh command always assumes that the PGN is preceeded by two zeros.

This command is only available when a CAN Protocol (A, B, or C) has been selected for SAE J1939 formatting. It returns an error if attempted under any other conditions. Note also that this version of the ELM327 only displays responses that match the criteria, not the requests that are asking for the information.

MP hhhhhh [Monitor for PGN hhhhhh]

This command is very similar to the MP hhhh command, but it extends the number of bytes provided by one, so that there is complete control over the PGN definition (it does not make the assumption that the Data Page bit is 0, as the MP hhhh command does). This allows for future expansion, should additional PGNs be defined with the Data Page bit set. Note that only the Data Page bit is relevant in the extra byte - the Reserved and Priority bits are ignored.

MR hh [Monitor for Receiver hh]

This command is very similar to the AT MA command except that it will only display messages that were sent to the hex address given by hh. These are messages which are found to 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, as with the MA command.

Note that if this command is used with CAN protocols, and if the CAN filter and/or mask were previously set (with CF, CM or CRA), then the MR command will over-write the previous values for these bits only - the others will remain unchanged. As an example, if the receive address has been set with CRA 4B0, and then you send MR 02, the 02 will replace the 4, and the CAN masks/filters will only allow IDs that are equal to 2B0. This is often not what is desired - you may want to reset the masks and filters (with AT AR) first.

As with the AT MA command, this command begins by performing an internal Protocol Close. Subsequent OBD requests may show "SEARCHING" or "BUS INIT", etc. messages when the protocol is reactivated.

MT hh [Monitor for Transmitter hh]

This command is also very similar to the AT MA command, except that it will only display messages that were sent by the transmitter with the hex address given by hh. These are messages which are found to have that value in the third byte of a traditional three byte OBD header, or in bits 0 to 7 for CAN IDs. As with

AT Command Descriptions (continued)

the MA and MR monitoring modes, any RS232 activity (single character) aborts the monitoring.

Note that if this command is used with CAN protocols, and if the CAN filter and/or mask were previously set (with CF, CM or CRA), then the MT command will over-write the previous values for these bits only - the others will remain unchanged. As an example, if the receive address has been set with CRA 4B0, and then you send MT 20, the 20 will replace the B0, and the CAN masks/filters will only allow IDs that are equal to 420. This is often not what is desired - you may want to reset the masks and filters (with AT AR) first.

As with the AT MA command, this command begins by performing an internal Protocol Close. Subsequent OBD requests may show “SEARCHING” or “BUS INIT”, etc. messages when the protocol is reactivated.

NL [Normal Length messages]

Setting the NL mode on forces all sends and receives to be limited to the standard seven data bytes in length, similar to the other ELM32x OBD ICs. To allow longer messages, use the AL command.

Beginning with v1.2, the ELM327 does not require a change to AL to allow longer message lengths for the KWP protocols to be received (as determined by the header length values). You can simply leave the IC set to the default setting of NL, and all of the received bytes will be shown.

PC [Protocol Close]

There may be occasions where it is desirable to stop (deactivate) a protocol. Perhaps you are not using the automatic protocol finding, and wish to manually activate and deactivate protocols. Perhaps you wish to stop the sending of idle (wakeup) messages, or have another reason. The PC command is used in these cases to force a protocol to close.

PP hh OFF [turn Prog. Parameter hh OFF]

This command disables Programmable Parameter number hh. Any value assigned using the PP hh SV command will no longer be used, and the factory default setting will once again be in effect. The actual time when the new value for this parameter becomes effective is determined by its type. Refer to the Programmable Parameters section (page 45) for more information on the types.

Note that ‘PP FF OFF’ is a special command that disables all of the Programmable Parameters, as if you had entered PP OFF for every possible one.

It is possible to alter some of the Programmable Parameters so that it may be difficult, or even impossible, to communicate with the ELM327. If this occurs, there is a hardware means of resetting all of the Programmable Parameters at once. Connect a jumper from circuit common to pin 28, holding it there while powering up the ELM327 circuit. Hold it in position until you see the RS232 Receive LED begin to flash (which indicates that all of the PPs have been turned off). At this point, remove the jumper to allow the IC to perform a normal startup. Note that a reset of the PPs occurs quite quickly – if you are holding the jumper on for more than a few seconds and do not see the RS232 receive light flashing, remove the jumper and try again – there may be a problem with your connection. This feature is only available beginning with v1.2, and is not a provided with any earlier versions of the ELM327 IC.

PP hh ON [turn Programmable Parameter hh ON]

This command enables Programmable Parameter number hh. Once enabled, any value assigned using the PP hh SV command will be used where the factory default value was before. (All of the programmable parameter values are set to their default values at the factory, so enabling a programmable parameter before assigning a value to it will not cause problems.) The actual time when the value for this parameter becomes effective is determined by its type. Refer to the Programmable Parameters section (page 45) for more information on the types.

Note that ‘PP FF ON’ is a special command that enables all of the Programmable Parameters at the same time.

PP xx SV yy [Prog. Param. xx: Set the Value to yy]

A value is assigned to a Programmable Parameter using this command. The system will not be able to use this new value until the Programmable Parameter has been enabled, however.

PPS [Programmable Parameter Summary]

The complete range of current Programmable Parameters are displayed with this command (even

AT Command Descriptions (continued)

those not yet implemented). Each is shown as a PP number followed by a colon and the value that is assigned to it. This is followed by a single digit – either 'N' or 'F' to show that it is ON (enabled), or OFF (disabled), respectively. See the Programmable Parameters section for a more complete discussion.

R0 and **R1** [Responses off (0) or on (1)]

These commands control the ELM327's automatic receive (and display) of the messages returned by the vehicle. If responses have been turned off, the IC 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 ELM327 does not print anything to say that the send was successful, but you will see a message if it was not).

R0 may be useful to send commands blindly when using the IC for a non-OBD network application, or when simulating an ECU in a learning environment. It is not recommended that this option normally be used, however, as the vehicle may have difficulty if it is expecting an acknowledgement byte and never receives one.

An R0 setting will always override any number of responses digit that is provided with an OBD request. The default setting is R1, or responses on.

RA hh [set the Receive Address to hh]

Depending on the application, users may wish to manually set the address to which the ELM327 will respond. Issuing this command will turn off the AR mode, and force the IC to only accept responses addressed to hh. Use caution with this setting, as depending on what you set it to, you may end up accepting (acknowledging with an IFR) a message that was actually meant for another module. To turn off the RA filtering, simply send AT AR.

This command is not very effective for use with the CAN protocols, as it only monitors for one portion of the ID bits, and that is not likely enough for most CAN applications - the CRA command may be a better choice. Also, this command has no effect on the addresses used by the J1939 protocols, as the J1939 routines derive them from the header values, as required by the SAE standard.

This RA command is exactly the same as the SR command, and can be used interchangeably with it.

RTR [send an RTR message]

This command causes a special "Remote Frame" CAN message to be sent. This type of message has no data bytes, and has its Remote Transmission Request (RTR) bit set. The headers and filters will remain as previously set (ie the ELM327 does not make any assumptions as to what format a response may have), so adjustments may need to be made to the mask and filter. This command must be used with an active CAN protocol, and will return an error if the protocol is not. Note that the CAF1 setting normally eliminates the display of all RTRs, so if you are monitoring messages and want to see the RTRs, you will have to turn off formatting, or turn the headers on.

The ELM327 treats an RTR just like any other message send, and will wait for a response from the vehicle (unless AT R0 has been chosen).

RV [Read the input Voltage]

This initiaties the reading of the voltage present at pin 2, and the conversion of it to a decimal voltage. By default, it is assumed that the input is connected to the voltage to be measured through a 47KΩ and 10KΩ resistor divider (with the 10KΩ connected from pin 2 to Vss), and that the ELM327 supply is a nominal 5V. This will allow for the measurement of input voltages up to about 28V, with an uncalibrated accuracy of typically about 2%.

S0 and **S1** [printing of Spaces off (0) or on (1)]

These commands control whether or not space characters are inserted in the ECU response.

The ELM327 normally reports ECU responses as a series of hex characters that are separated by space characters (to improve readability), but messages can be transferred much more quickly if every third byte (the space) is removed. While this makes the message less readable for humans, it can provide significant improvements for computer processing of the data. By default, spaces are on (S1), and space characters are inserted in every response.

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

Entering CAN 11 bit ID words (headers) normally requires that extra leading zeros be added (eg. AT SH 00 07 DF), but this command simplifies doing so. The AT SH xyz command accepts a three digit argument, takes only the right-most 11 bits from that, adds

AT Command Descriptions (continued)

leading zeros, and stores the result in the header storage locations for you. As an example, AT SH 7DF is a valid command, and is quite useful for working with 11 bit CAN systems. It actually results in the header bytes being internally stored as 00 07 DF.

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

This command allows the user to manually control the values that are sent as the three header bytes in a message. These bytes are normally assigned values for you (and are not required to be adjusted), but there may be occasions when it is desirable to change them (particularly if experimenting with physical addressing).

The header bytes are defined with hexadecimal 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 D, WS, or Z commands.

If new values for header bytes are set before the vehicle protocol has been determined, and if the search is not set for fully automatic (ie other than protocol 0), these new values will be used for the header bytes of the first request to the vehicle. If that first request should fail to obtain a response, and if the automatic search is enabled, the ELM327 will then continue to search for a protocol using default values for the header bytes. Once a valid protocol is found, the header bytes will revert to the values assigned with the AT SH command.

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 (and any extra bits assigned will be ignored). The additional 5 bits needed for a 29 bit system are set with the AT CP command.

If assigning header values for the KWP protocols (4 and 5), care must be taken when setting the first header byte (xx) value. The ELM327 will always insert the number of data bytes for you, but how it is done depends on the values that you assign to this byte. If the second digit of this first header byte is anything other than 0 (zero), the ELM327 assumes that you wish to have the length value inserted in that first byte when sending. In other words, providing a length value in the first header byte tells the ELM327 that you wish to use a traditional 3 byte header, where the length is stored in the first byte of the header.

If you provide a value of 0 for the second digit of the first header byte, the ELM327 will assume that you wish that value to remain as 0, and that you want to have a fourth header (length) byte inserted into the message. This is contrary to the ISO 14230-4 OBD standard, but it is in use by many KWP2000 systems for (non-OBD) data transfer, so may be useful when experimenting. Support for 4 byte KWP headers was added with v1.2 of the ELM327 IC, and is not available in previous versions.

SP h [Set Protocol to h]

This command is used to set the ELM327 for operation using the protocol specified by 'h', and to also save it as the new default. Note that the protocol will be saved no matter what the AT M0/M1 setting is.

The currently 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)
A - SAE J1939 CAN (29 bit ID, 250* Kbaud)
B - USER1 CAN (11* bit ID, 125* Kbaud)
C - USER2 CAN (11* bit ID, 50* Kbaud)

* default settings (user adjustable)

The first protocol shown (0) is a convenient way of telling the ELM327 that the vehicle's protocol is not known, and that it should perform a search. It causes the ELM327 to try all protocols if necessary, looking for one that can be initiated correctly. When a valid protocol is found, and the memory function is enabled, that protocol will then be remembered, and will become the new default setting. When saved like this, the automatic mode searching will still be enabled, and the next time the ELM327 fails to connect to the saved protocol, it will again search all protocols for another valid one. Note that some vehicles respond to more than one protocol - during a search, you may see more than one type of response.

AT Command Descriptions (continued)

ELM327 users often use the AT SP 0 command to reset the search protocol before starting (or restarting) a connection. This works well, but as with any Set Protocol command, it involves a write to EEPROM, and an unnecessary delay (of about 30 msec) while the write occurs. Beginning with v1.3 of the ELM327, a write to EEPROM will no longer be performed for an SP 0 (or an SP A0, or SP 0A) command, but it will still reset the protocol to 0 for you. If you really want to change what is stored in the internal EEPROM, you must now use the new AT SP 00 command.

If another protocol (other than 0) is selected with this command (eg. AT SP 3), that protocol will become the default, and will be the only protocol used by the ELM327. Failure to initiate a connection in this situation will result in familiar responses such as 'BUS INIT: ...ERROR', and no other protocols will be attempted. This is a useful setting if you know that your vehicle(s) only require the one protocol.

SP Ah [Set Protocol to Auto, h]

This variation of the SP command allows you to choose a starting (default) protocol, while still retaining the ability to automatically search for a valid protocol on a failure to connect. For example, if your vehicle is ISO 9141-2, but you want to occasionally use the ELM327 circuit on other vehicles, you might use the AT SP A3 command, so that the first protocol tried will then be yours (3), but it will also automatically search for other protocols. Don't forget to disable the memory function if doing this, or your neighbour's protocol could become your new default.

As for SP h, an SP Ah will save the protocol information even if the memory option is off (except for SP A0 and SP 0A which as of v1.3 no longer cause a write - if you need to, use SP 00). Note that the 'A' can come before or after the h, so AT SP A3 can also be entered as AT SP 3A.

SR hh [Set the Receive address to hh]

Depending on the application, users may wish to manually set the address to which the ELM327 will respond. Issuing this command will turn off the AR mode, and force the IC to only accept responses addressed to hh. Use caution with this setting, as depending on what you set it to, you may end up accepting (acknowledging with an IFR) a message that was actually meant for another module. To turn off the SR filtering, simply send AT AR.

This command is not very effective for use with the CAN protocols, as it only monitors for one portion of the ID bits, and that is not likely enough for most CAN applications - the CRA command may be a better choice. Also, this command has no effect on the addresses used by the J1939 protocols, as the J1939 routines derive them from the header values, as required by the SAE standard.

This SR command is exactly the same as the RA command, and can be used interchangeably with it.

ST hh [Set Timeout to hh]

After sending a request, the ELM327 waits a preset time for a response before it can declare that there was 'NO DATA' received from the vehicle. The same timer setting is also used after a response has been received, while waiting to see if more are coming. The AT ST command allows this timer to be adjusted, in increments of 4 msec.

When Adaptive Timing is enabled, the AT ST time sets the maximum time that is to be allowed, even if the adaptive algorithm determines that the setting should be higher. In most circumstances, it is best to leave the AT ST time at the default setting when using adaptive timing.

The ST timer is set to 32 by default (giving a time of approximately 200 msec), but this value can be adjusted by changing PP 03. Note that a value of 00 does not result in a time of 0 msec – it will restore the timer to the default value.

SW hh [Set Wakeup to hh]

Once a data connection has been made, some vehicles require that there be data flow every few seconds, or the connection may time out and 'go to sleep.' The ELM327 will automatically generate periodic 'wakeup' messages in order to maintain this connection, whenever the user is not requesting any data. (Currently, only protocols 3, 4, and 5 generate these messages.) The replies to these messages are always ignored, and are not visible to the user.

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. The maximum possible time delay of just over 5 seconds thus occurs when a value of FF (decimal 255) is used. The default setting provides a nominal delay of 3 seconds between messages.

AT Command Descriptions (continued)

Note that the value 00 (zero) is treated as a very special case, and must be used with caution, as it will stop all periodic messages. This is provided as it may be convenient in certain circumstances. Issuing AT SW 00 will not change a prior setting for the time between wakeup messages, should the protocol be re-initialized.

TP h [Try Protocol h]

This command is identical to the SP command, except that the protocol that you select is not immediately saved in internal memory, so does not change the default setting. Note that if the memory function is enabled (AT M1), and this new protocol that you are trying is found to be valid, that protocol will then be stored in memory as the new default.

TP Ah [Try Protocol h with Auto]

This command is very similar to the AT TP command above, except that if the protocol that is tried should fail to initialize, the ELM327 will then automatically sequence through all of the protocols, attempting to connect to one of them.

V0 and **V1** [Variable data lengths off (0) or on (1)]

These commands modify the current CAN protocol settings to allow the sending of variable data length messages, just as bit 6 of PP 2C and PP 2E do for protocols B and C. This allows experimenting with variable data length messages for any of the CAN protocols (not just B and C). The V1 command will always override any protocol setting, and force a variable data length message. The default setting is V0, providing data lengths as determined by the protocol.

WM [1 to 6 bytes] [set Wakeup Message to…]

This command allows the user to override the default settings for the wakeup messages (sometimes known as the 'periodic idle' messages). Simply provide the bytes that you wish to have sent (one to six of them), and the ELM327 will then send them as required, at the rate determined by the AT SW setting. Note that you do not have to add a checksum byte to the data – the ELM327 calculates the value and adds it for you.

WS [Warm Start]

This command causes the ELM327 to perform a complete reset which is very similar to the AT Z command, but does not include the power on LED test. Users may find this a convenient means to quickly 'start over' without having the extra delay of the AT Z command.

If using variable RS232 baud rates (ie AT BRD commands), it is preferred that you reset the IC using this command rather than AT Z, as AT WS will not affect the chosen RS232 baud rate.

Z [reset all]

This command causes the chip 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 chip will be put in the idle state, waiting for characters on the RS232 bus. Any baud rate that was set with the AT BRD command will be lost, and the ELM327 will return to the default baud rate setting.

@1 [display the device description]

This command causes the internal device description string to be printed. The default text is "OBDII to RS232 Interpreter".

@2 [display the device identifier]

The device identifier that was recorded with the @3 command is displayed with an AT @2 command. All 12 characters and a terminating carriage return will be sent in response, if they have been defined. If no identifier has been set, the @2 command returns an error response ('?'). The identifier may be useful for storing product codes, production dates, serial numbers, or other such codes.

@3 ccccccccccc [store the device identifier]

This command is used to set the device identifier code. Exactly 12 characters must be sent, and once written to memory, they can not be changed (ie it is 'write once, read many'). The characters sent must be printable (ascii character values 00x21 to 0x5F inclusive).

Reading the Battery Voltage

Before proceding to the OBD Commands, we will show an example of how to use an AT Command. We will assume that you have built (or purchased) a circuit which is similar to that of Figure 9 in the Example Applications section (page 56). This circuit provides a connection to read the vehicle's battery voltage, which many will find very useful.

If you look in the AT Command list, you will see there is one command that is listed as RV [Read the input Voltage]. This is the command which you will need to use. First, be sure that the prompt character is shown (that is the '>' character), then simply enter 'AT' followed by RV, and press return (or enter):

```
>AT RV
```

Note that we used upper case characters for this request, but it was not required, as the ELM327 will accept upper case (AT RV) as well as lower case (at rv) or any combination of these (At rV). It does not matter if you insert space characters (' ') between the 'AT' and the 'RV' either, as they will be ignored by the ELM327.

A typical reposnse to this command will show a voltage reading, followed by another prompt character:

```
12.6V

>
```

The accuracy of this reading depends on several factors. As shipped from the factory, the ELM327 voltage reading circuitry will typically be accurate to about 2%. For many, this is all that is needed. Some people may want to calibrate the circuitry for more accurate readings, however, so we have provided a special 'Calibrate Voltage' command for this.

To change the internal calibration constants, you will need to know the actual battery voltage to more accuracy than the ELM327 shows. Many quality digital multimeters can do this, but you should verify the accuracy before making a change.

For this example, let us assume that you have connected your multimeter, and you find that it reads 12.47V. The ELM327 is a little high at 12.6V, and you would like the ELM327 to read the same as your meter. Simply calibrate the ELM327 to that voltage using the CV command:

```
>AT CV 1247
OK
```

Note that you should not provide a decimal point in the CV value, as the ELM327 knows that it should be between the second and the third digits.

At this point, the internal calibration values have been changed (ie. written to EEPROM), and the ELM327 now knows that the voltage at the input is actually 12.47V. To verify that the changes have taken place, simply read the voltage again:

```
>AT RV
12.5V

>
```

The ELM327 always rounds off the measurement to one decimal place, so the 12.47V actually appears as 12.5V (but the second decimal place is maintained internally for accuracy and is used in the calculations). There is currently no way to restore the ELM327 to the factory setting once you have sent a CV command, so be careful while experimenting with it.

The ELM327 can be calibrated with any reference voltage that you have available, but note that the CV command always expects to receive four characters representing the voltage at the input. If you had used a 9V battery for your reference, and it is actually 9.32V, then you must add a leading zero to the actual voltage when calibrating the IC:

```
>AT CV 0932
OK

>
```

The other AT Commands are used in the same manner. Simply type the letters A and T, then follow that with the command you want to send, then any arguments that are required for that command, and press return (or enter, depending on your keyboard). You can always insert space characters as often as you wish if it improves the readability for you, as they are ignored by the ELM327.

OBD Commands

If the bytes that you send to the ELM327 do not begin with the letters ‘A’ and ‘T’, they are assumed to be OBD commands for the vehicle. Each pair of ASCII bytes will be tested to ensure that they are valid hexadecimal digits, and will then be combined into data bytes for transmitting to the vehicle.

OBD commands are actually sent to the vehicle embedded in a data packet. Most standards require that three header bytes and an error checksum byte be included with every OBD message, and the ELM327 adds these extra bytes to your command bytes automatically. The initial (default) values for these extra bytes are usually appropriate for most requests, but if you wish to change them, there is a mechanism to do so (see the “Setting the Headers” section).

Most OBD commands are only one or two bytes in length, but some can be longer. The ELM327 will limit the number of bytes that can be sent to the maximum number allowed by the standards (seven bytes or 14 hexadecimal digits). Attempts to send more bytes will result in a syntax error – the entire command is then ignored and a single question mark printed.

Hexadecimal digits are used for all of the data exchange with the ELM327 because it is the data format used most often in the OBD standards. Most mode request listings use hexadecimal notation, and it is the format most frequently used when results are shown. With a little practice, it should not be very difficult to deal in hex numbers, but some people may want to use a table such as Figure 1, or keep a calculator nearby. All users will be required to manipulate the results in some way, though – combining bytes and dividing by 4 to obtain rpm, dividing by 2 to obtain degrees of advance, etc., and may find a software front-end to be of help.

As an example of sending a command to the vehicle, assume that A6 (or decimal 166) is the command that is required to be sent. In this case, the user would type the letter A, then the number 6, then would press the return key. These three characters would be sent to the ELM327 by way of the RS232 port. The ELM327 would store the characters as they are received, and when the third character (the carriage return) was received, would begin to assess the other two. It would see that they are both valid hex digits, and would convert them to a one byte value (the decimal value is 166). The header bytes and a checksum byte would then be added, and a total of five bytes would typically be sent to the vehicle. Note that the carriage return character is only a signal to the ELM327, and is not sent on to the vehicle.

After sending the command, the ELM327 listens on the OBD bus for replies, looking for ones that are directed to it. If a message address matches, the received bytes will be sent on the RS232 port to the user, while messages received that do not have matching addresses will be ignored (but are often still available for viewing with the AT BD command).

The ELM327 will continue to wait for messages addressed to it until there are none found in the time that was set by the AT ST command. As long as messages continue to be received, the ELM327 will continue to reset this timer, and look for more. Note that the IC will always respond to a request with some reply, even if it is to say “NO DATA” (meaning that there were no messages found, or that some were found but they did not match the receive criteria).

Hexadecimal Number	Decimal Equivalent
0	0
1	1
2	2
3	3
4	4
5	5
6	6
7	7
8	8
9	9
A	10
B	11
C	12
D	13
E	14
F	15

Figure 1. Hex to Decimal Conversion

Talking to the Vehicle

The OBD standards specify that each request that is sent to the vehicle must adhere to a set format. The first byte sent (known as the 'mode') describes the type of data being requested, while the second byte (and possibly a third or more) specify the actual information that is required. The bytes which follow after the mode byte are known as the 'parameter identification' or PID number bytes. The modes and PIDs are described in detail in documents such as the SAE J1979, or ISO 15031-5 standards, and may also be expanded on by the vehicle manufacturers.

Initially, SAE standard J1979 defined nine possible diagnostic test modes, which 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

Recent changes now define a tenth mode that may be used with the ISO 15765-4 (CAN) standard. This mode (0A) is used for permanent trouble codes.

Vehicles are not required to support all modes, and within modes, they are not required to support all possible PIDs (some of the first OBDII vehicles only supported a very small number). Within each mode, PID 00 is normally reserved to show which PIDs are supported by that mode, and can be used to determine which ones your vehicle supports.

Mode 01, PID 00 must be supported by all vehicles, and can be accessed as follows…

Ensure that your ELM327 interface is properly connected to the vehicle, and powered. Most vehicles will not respond without the ignition key in the ON position, so turn the ignition to on, but do not start the engine. At the prompt, issue the mode 01 PID 00 command:

```
>01 00
```

The first time the bus is accessed, you may see a bus initialization message, followed by a response, which might typically be as follows:

```
41 00 BE 1F B8 10
```

The 41 signifies a response from a mode 01 request (01 + 40 = 41), while the second number (00) simply repeats the PID number requested. A mode 02, request is answered with a 42, a mode 03 with a 43, etc. The next four bytes (BE, 1F, B8, and 10) represent the requested data, in this case a bit pattern showing the PIDs that are supported by this mode (1=supported, 0=not). Although this information is not very useful for the casual user, it does prove that the connection is working.

Another example requests the current engine coolant temperature (ECT). Coolant temperature is PID 05 in mode 01, and can be requested as follows:

```
>01 05
```

The response will be of the form:

```
41 05 7B
```

The 41 05 shows that this is a response to a mode 1 request for PID 05, while the 7B is the desired data. Converting the hexadecimal 7B to decimal, one gets 7 x 16 + 11 = 123. This represents the current temperature in degrees Celsius, but with the zero offset to allow for subzero temperatures. To convert to the actual coolant temperature, you need to subtract 40 from the value obtained. In this case, then, the coolant temperature is 123 - 40 or 83°C.

A final example shows a request for the engine rpm. This is PID 0C of mode 01, so at the prompt type:

```
>01 0C
```

A typical response would be:

```
41 0C 1A F8
```

The returned value (1A F8) is actually a two byte value that must be converted to a decimal value to be useful. Converting it, we get a value of 6904, which seems to be a very high value for engine rpm. That is because rpm is sent in increments of 1/4 rpm! To convert to the actual engine speed, we need to divide the 6904 by 4. In this case, then, the rpm is 1726, which is much more reasonable.

When the ELM327 sends a request, it also sets a timer that limits the amount of time that the IC will wait for a response (you can adjust this timer with the AT ST command). If no response is obtained during this time, you will see a NO DATA message, and if a response is received, it will be printed as we have just seen.

Talking to the Vehicle (continued)

After the first response has been received and printed, the timer is reset, and the ELM327 waits to see if another response is coming. If nothing does arrive, you will not see a NO DATA response however, as there has been data returned already. On such a timeout, the IC will simply print a prompt character, and wait for the next command from the PC.

This timeout approach means that the ELM327 may take almost twice as long as it needs to, in order to obtain one single answer, then return to the prompt. Unfortunately, there was little that could be done about this with the earlier versions of this IC.

Version 1.3 of the ELM327 introduces one way to speed up the retrieval of information, if you know how many responses to expect. You can now tell the ELM327 how many responses to obtain before returning immediately to the prompt state. This is done by adding a single hex digit after the OBD request bytes - the value of the digit providing the maximum number of responses to obtain. As an example, if you know that there is only one response coming for the engine temperature request that was previously discussed, you can now send:

```
>01 05 1
```

and the ELM327 will return immediately after obtaining only one response. It still sets the internal timer after sending the request, but it only waits for one response - it does not set the timer and wait for a second response, which may save considerable time (the default time without adaptive timing being enabled is 200 msec).

The number of responses that you provide can be any single hex digit, of value 0 to F. It is not required to be provided (the old way of doing it is fine - it's just a little slower). Note that the AT R0 setting will override any number provided with the OBD request. If you try to obtain more responses than the ECU can provide, the ELM327 will time out as previously, and there is no harm done.

A final example is the request for a vehicle serial number, which is discussed in a coming section. The mode for the serial number is 09, while the PID is 02. If you know that the response is to be 5 lines long, time can be saved by sending:

```
>09 02 5
```

Before using this new method of data retrieval, be sure to go to the trouble of learning how many responses are normally sent by your vehicle in response to a query. If you should request too few, ignoring the others, you may cause undo congestion of the vehicle data bus as a result. For example, the J1850 PWM protocol requires that every response be acknowledged with what is known as an 'in frame response' (IFR). If this IFR is not received by the vehicle, it will try to resend the data (many times), hoping to eventually receive an IFR. If you ask for only one response, but the vehicle is trying to send 5, you may cause problems.

Clearly, there needs to be a way to keep track of how many responses to expect, and then make the proper requests based on this information. For a person trying to obtain a few trouble codes, the savings are not worth the trouble (it's likely easiest to use the old way to make a request), but for a programmer trying to obtain vehicle information as quickly as possible, the time taken is most likely very worthwhile.

We must present another warning here to be very cautious with how you use this method of querying the vehicle. One direct benefit of the ELM327 returning more quickly from making a request is that the next request can then begin more quickly. Be aware that early versions of the SAE J1979 standard forbid sending requests more frequently than every 100 msec, however, and it was only with the APR2002 update that scan tools were finally allowed to send the next request without delay. Furthermore, a request can only be sent without delay if it is determined that all the responses have been received. If you are connected to an older vehicle, you should use the old method of making requests.

Hopefully this has shown how typical requests are made using the ELM327. It has not been meant to be a definitive guide on modes and PIDs – this information can be obtained from the manufacturer of your vehicle, the SAE (http://www.sae.org/), from ISO (http://www.iso.org/), or from various other sources on the web.

Interpreting Trouble Codes

Likely the most common use that the ELM327 will be put to is in obtaining the current Diagnostic Trouble Codes (or DTCs). Minimally, this requires that a mode 03 request be made, but first one should determine how many trouble codes are presently stored. This is done with a mode 01 PID 01 request as follows:

```
>01 01
```

To which a typical response might be:

```
41 01 81 07 65 04
```

The 41 01 signifies a response to the request, and the next data byte (81) is the number of current trouble codes. Clearly there would not be 81 (hex) or 129 (decimal) trouble codes present if the vehicle is at all operational. In fact, this byte does double duty, with the most significant bit being used to indicate that the malfunction indicator lamp (MIL, or 'Check Engine Light') has been turned on by one of this module's codes (if there are more than one), while the other 7 bits of this byte provide the actual number of stored trouble codes. In order to calculate the number of stored codes when the MIL is on, simply subtract 128 (or 80 hex) from the number.

The above response then indicates that there is one stored code, and it was the one that set the Check Engine Lamp or MIL on. The remaining bytes in the response provide information on the types of tests supported by that particular module (see the SAE document J1979 for further information).

In this instance, there was only one line to the response, but if there were codes stored in other modules, they each could have provided a line of response. To determine which module is reporting the trouble code, one would have to turn the headers on (AT H1) and then look at the third byte of the three byte header for the address of the module that sent the information.

Having determined the number of codes stored, the next step is to request the actual trouble codes with a mode 03 request (there is no PID needed):

```
>03
```

A response to this could be:

```
43 01 33 00 00 00 00
```

The '43' in the above response simply indicates that this is a response to a mode 03 request. The other 6 bytes in the response have to be read in pairs to show the trouble codes (the above would be interpreted as 0133, 0000, and 0000). Note that the response has been padded with 00's as required by the SAE standard for this mode – the 0000's do not represent actual trouble codes.

As was the case when requesting the number of stored codes, the most significant bits of each trouble code also contain additional information. It is easiest to use the following table to interpret the extra bits in the first digit as follows:

If the first hex digit received is this,	Replace it with these two characters	
0	P0	Powertrain Codes - SAE defined
1	P1	" " - manufacturer defined
2	P2	" " - SAE defined
3	P3	" " - jointly defined
4	C0	Chassis Codes - SAE defined
5	C1	" " - manufacturer defined
6	C2	" " - manufacturer defined
7	C3	" " - reserved for future
8	B0	Body Codes - SAE defined
9	B1	" " - manufacturer defined
A	B2	" " - manufacturer defined
B	B3	" " - reserved for future
C	U0	Network Codes - SAE defined
D	U1	" " - manufacturer defined
E	U2	" " - manufacturer defined
F	U3	" " - reserved for future

Taking the example trouble code (0133), the first digit (0) would then be replaced with P0, and the 0133 reported would become P0133 (which is the code for an 'oxygen sensor circuit slow response'). Note that the ISO 15765-4 (CAN) protocol is very similar, but it adds an extra data byte (in the second position), showing how many data items (DTCs) are to follow.

To provide a few further examples, if the received code was D016, you would replace the D with U1, and the resulting trouble code would be U1016. Similarly, 1131 received would actually be for the code P1131.

Resetting Trouble Codes

The ELM327 is quite capable of resetting diagnostic trouble codes, as this only requires issuing a mode 04 command. The consequences should always be considered before sending it, however, as more than the MIL (or 'Check Engine Light') will be reset. In fact, issuing a mode 04 will:

- reset the number of trouble codes
- erase any diagnostic trouble codes
- erase any stored freeze frame data
- erase the DTC that initiated the freeze frame
- erase all oxygen sensor test data
- erase mode 06 and 07 information

Clearing of all of this data is not unique to the ELM327 – it occurs whenever any scan tool is used to reset the codes. The biggest problem with losing this data is that your vehicle may run poorly for a short time, while it performs a recalibration.

To avoid inadvertently erasing stored information, the SAE specifies that scan tools must verify that a mode 04 is intended ("Are you sure?") before actually sending it to the vehicle, as all trouble code information is immediately lost when the mode is sent. Remember that the ELM327 does not monitor the content of the messages, so it will not know to ask for confirmation of the mode request – this would have to be the duty of a software interface if one is written.

As stated, to actually erase diagnostic trouble codes, one need only issue a mode 04 command. A response of 44 from the vehicle indicates that the mode request has been carried out, the information erased, and the MIL turned off. Some vehicles may require a special condition to occur (eg. the ignition on but the engine not running) before they will respond to a mode 04 command.

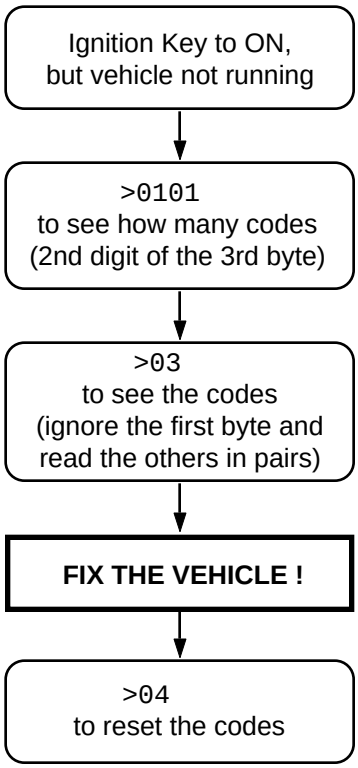
That is all there is to clearing trouble codes. Once again, do not accidentally send the 04 code!

Quick Guide for Reading Trouble Codes

If you do not use your ELM327 for some time, this entire data sheet may seem like quite a bit to review when your 'Check Engine' light eventually comes on, and you just want to know why. We offer this section as a quick guide to the basics that you will need.

To get started, connect the ELM327 circuit to your PC or PDA and communicate with it using a terminal program such as HyperTerminal, ZTerm, ptelnet, or a similar program. It should normally be set to either 9600 or 38400 baud, with 8 data bits, and no parity or handshaking.

The chart at the right provides a quick procedure on what to do next:



Bus Initiation

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) initiation process, while ISO 14230 allows for both a slow method, and a faster alternative.

The ELM327 will perform this bus initiation for you, but not until your request needs to be sent. If the bus initiation occurs during an automatic search, you will not see any status reporting, but if you have the Auto option off (and are set to protocols 3, 4, or 5), 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 the dots. 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 vehicle's key to 'ON' before attempting to talk to the vehicle.)

Once the bus has been initiated, communications must take place regularly (typically at least once every five seconds), or the bus will revert to a low-power 'sleep' mode. If you are not sending data requests often enough, the ELM327 will generate requests for you to ensure that the bus stays 'awake'. You will never see the responses to these, but you may see the transmit LED flash periodically when these are being sent.

By default, the ELM327 ensures that these 'wakeup' or 'idle' messages are sent every 3 seconds, but this is adjustable with the AT SW command. The contents of the wakeup message are also user programmable with the AT WM command, if you should wish to change them. Users generally do not have to change either of the above though, as the default settings work well with almost all systems.

Wakeup Messages

After an ISO 9141 or ISO 14230 connection has been established, there needs to be periodic data transfers in order to maintain that connection, and prevent it from 'going to sleep.' If normal requests and responses are being sent, that is usually sufficient, but the ELM327 occasionally has to create its own messages to prevent the connecton from timing out.

We term these periodic messages that are created the 'Wakeup Messages', as they keep the connection alive, and prevent the circuitry from going back to an idle or sleep mode. (Some texts refer to these messages simply as 'idle messages.') The ELM327 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 'blips' created when the ELM327 sends one. If you are curious as to the actual contents of the messages, you can then perform a Buffer Dump to see the bytes. Note that the ELM327 never obtains or prints a response to any of the wakeup messages.

The standards state that if there is no activity at least every five seconds, the data connection may close. To ensure that this does not happen, and to provide some margin, the ELM327 will send a wakeup message after three seconds of inactivity. This time interval is fully programmable, should you prefer a different setting (see the AT SW command).

As with the ELM323, the ELM327 allows users to change the actual wakeup message that is sent. To do so, simply send the ELM327 a Wakeup Message command, telling it what you wish the message to be changed to. For example, if you would like to send the data bytes 44 55 with the header bytes set to 11 22 33, simply send the command:

```
>AT WM 11 22 33 44 55
```

and from that point forward, every wakeup message that the ELM327 sends will be as shown. You should not provide a checksum byte, as it will be automatically added for you.

You can change these as often as you want, the only restriction being that every time you do, you must provide the complete message – the header bytes and the data bytes (the current version of the ELM327 only allows for messages of one to six bytes in length).

Selecting Protocols

The ELM327 supports several different OBD protocols (see Figure 2, at right). As a user, you may never have to choose which one it should use (since the factory settings cause an automatic search to be performed for you), but while experimenting, you may want to specify which protocol is to be used.

For example, if you know that your vehicle uses the SAE J1850 VPW protocol, you may want the ELM327 to use only that protocol, and no others. If that is what you want, simply determine the protocol number (from Figure 2), then use the 'Set Protocol' AT Command as follows:

```
>AT SP 2
OK
```

From this point on, the default protocol (used after every power-up or AT D command) will be protocol 2 (or whichever one that you have chosen). Verify this by asking the ELM327 to describe the protocol:

```
>AT DP
SAE J1850 VPW
```

Now what happens if your friend has a vehicle that uses ISO 9141-2? How do you now use the ELM327 interface for that vehicle, if it is set for J1850?

One possibility is to change your protocol selection to allow for the automatic searching for another protocol, on failure of the current one. This is done by putting an 'A' with the protocol number, as follows:

```
>AT SP A2
OK

>AT DP
AUTO, SAE J1850 VPW
```

Now, the ELM327 will begin by trying protocol 2, but will then automatically begin searching for another protocol should an attempt to connect with protocol 2 fail (as would happen when you try to connect to the friend's vehicle).

The Set Protocol commands cause an immediate write to the internal EEPROM, before even attempting to connect to the vehicle. This write is time-consuming, affects the setting for the next powerup, and may not actually be appropriate, if the protocol selected is not correct for the vehicle. To allow a test before a write occurs, the ELM327 offers one other command - the Try Protocol (TP) command.

Try Protocol is very similar to Set Protocol. It is used in exactly the same way as the AT SP command, the only difference being that a write to internal memory will only occur after a valid protocol is found, and only if the memory function is enabled (M0/M1). For the previous example, all that needs to be sent is:

```
>AT TP A2
OK
```

Many times, it is very difficult to even guess at a protocol to try first. In these cases, it is best to simply let the ELM327 decide what to use. This is done by telling it to use protocol 0 (with either the SP or the TP commands).

To have the ELM327 automatically search for a protocol to use, simply send:

```
>AT SP 0
OK
```

and when the next OBD command is to be sent, the ELM327 will automatically look for one that responds. You will see a "SEARCHING..." message, followed by a response, after which you can ask the ELM327 what protocol it found (by sending AT DP).

The first versions of the ELM327 used the SAE recommended search order (protocol 1, 2, 3, etc.), but recent versions of the IC modify the search order

Protocol	Description
0	Automatic
1	SAE J1850 PWM (41.6 Kbaud)
2	SAE J1850 VPW (10.4 Kbaud)
3	ISO 9141-2 (5 baud init)
4	ISO 14230-4 KWP (5 baud init)
5	ISO 14230-4 KWP (fast init)
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)
A	SAE J1939 CAN (29 bit ID, 250* Kbaud)
B	User1 CAN (11* bit ID, 125* Kbaud)
C	User2 CAN (11* bit ID, 50* Kbaud)

*user adjustable

Figure 2. ELM327 Protocol Numbers

Selecting Protocols (continued)

based on any active inputs that are present. If you need to follow the SAE J1978 order, you will need to select each protocol with the SP or TP commands.

The automatic search works well with OBDII systems, but may not be what you need if you are experimenting. During a search, the ELM327 ignores any headers that you have previously defined (since there is always a chance that your headers may not result in a response), and it uses the default OBD header values for each protocol. It will also use standard requests (ie 01 00) during the searches. If this is not what you want, the results may be a little frustrating.

To use your own header (and data) values when attempting to connect to an ECU, do not tell the ELM327 to use protocol 0. Instead, tell it to either use only your target protocol (ie. AT SP n), or else tell it to use yours with automatic searches allowed on failure (ie AT SP An). Then send your request, with headers assigned as required. The ELM327 will attempt to connect using your headers and your data, and only if it fails (and you have chosen the protocol with the AT SP An method) will it search using the standard OBD default values.

In general, 99% of all users find that enabling the memory (setting pin 5 to 5V) and choosing the 'Auto' option the first time (the easiest way is to say AT SP 0) works very well. After the initial search, the protocol used by your vehicle becomes the new default (so it is tried first every time), and if the interface is used on another vehicle, there is only a minor delay while it performs an automatic search.

OBD Message Formats

To this point we have only discussed the contents (data portion) of an OBD message, and made only passing mention of other parts such as headers and checksums, which all messages use to some extent.

On Board Diagnostics systems are designed to be very flexible, providing a means for several devices to communicate with one another. In order for messages to be sent between devices, it is necessary to add information describing the type of information being sent, the device that it is being sent to, and perhaps which device is doing the sending. Additionally, the importance of the message becomes a concern as well – crankshaft position information is certainly of considerably more importance to a running engine than a request for the number of trouble codes stored, or the vehicle serial number. So to convey importance, messages are also assigned a priority.

The information describing the priority, the intended recipient, and the transmitter are usually needed by the recipient even before they know the type of request that the message contains. To ensure that this information is obtained first, OBD systems transmit it at the start (or head) of the message. Since these bytes are at the head, they are usually referred to as header bytes. Figure 3 below shows a typical OBD message structure that is used by the SAE J1850, ISO 9141-2, and ISO 14230-4 standards. It uses 3 header bytes as shown, to provide details concerning the priority, the receiver, and the transmitter. Note that many texts refer to the receiver

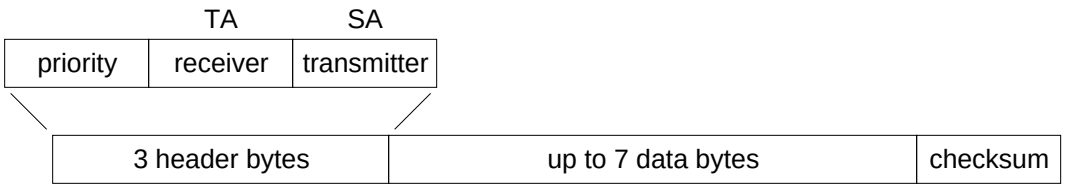


Figure 3. An OBD Message

OBD Message Formats (continued)

as the “Target Address” (TA), and the transmitter as the “Source Address” (SA).

Another concern when sending any message is that errors might occur in the transmission, and the received data may be falsely interpreted. To detect errors, the various protocols all provide some form of check on the received data. This may be as simple as a sum calculation (ie a ‘running total’ of byte values) that is sent at the end of a message. If the receiver also calculates a sum as bytes are received, then the two values can be compared and if they do not agree, the receiver will know that an error has occured. Since simple sums might not detect multiple errors, a more reliable (and more complicated) sum called a Cyclic Redundancy Check (or ‘CRC’) is often used. All of the protocols specify how errors are to be detected, and the various ways of handling them if they occur.

The OBD data bytes are thus normally encapsulated within a message, with ‘header’ bytes at the beginning, and a ‘checksum’ at the end. The J1850, ISO 9141-2, and ISO 14230-4 protocols all use essentially the same structure, with three header bytes, a maximum of seven data bytes and one checksum byte.

The ISO 15765-4 (CAN) protocol uses a very similar structure (see Figure 4, below), the main difference really only relating to the structure of the header. CAN header bytes are not referred to as header bytes – they are called ‘ID bits’ instead. The initial CAN standard defined the ID bits as being 11 in number, while the more recent CAN standard now allows for 11 or 29 bit IDs.

The ELM327 does not normally show any of these extra bytes unless you turn that feature on with the Headers On command (AT H1). Issuing that allows you to see the header bytes and the checksum byte (for the J1850, ISO 9141 and ISO 14230 protocols). For the CAN protocols, you will see the ID bits, and other items which are normally hidden such as the PCI byte for ISO 15765, or the data length codes (if they are enabled with PP 29, or AT D1). Note that the ELM327 does not display the checksum information for CAN systems, or the IFR bytes for J1850 systems.

It is not necessary to ever have to set these header byes, or to perform a checksum calculation, as the ELM327 will always do this for you. The header bytes are adjustable however, should you wish to experiment by using advanced techniques such as physical addressing. The next section provides a discussion on how to do this…

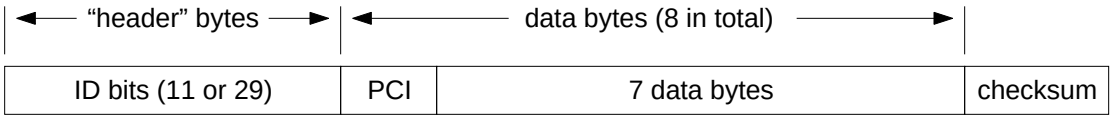


Figure 4. A CAN OBD Message

Setting the Headers

The emissions related diagnostic trouble codes that most people are familiar with are described in the SAE J1979 standard (ISO15031-5). They 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.

Accessing most OBDII diagnostics information requires that requests be made to what is known as a 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). In addition, 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 bytes 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. SAE standard J2178 also tells you that for your J1850 VPW vehicle, a priority byte of A8 is appropriate. Finally, knowing that a scan tool is normally address F1, you have enough information to specify the three header bytes (A8 48 and F1). To tell the ELM327 to use these new header bytes, all you need is the Set Header command:

```
>AT SH A8 48 F1
OK
```

The three header bytes assigned in this manner will stay in effect until changed by the next AT SH command, a reset, or an AT D.

Having set the header bytes, you now need only send the secondary ID for fluid temperature (10) at the prompt. If the display of headers is turned off, the conversation could look like this:

```
>10
10 2E
```

The first byte in the response echos the request, as usual, while the data that we requested is the 2E byte. You may find that some requests, being of a low priority, may not be answered immediately, possibly causing a "NO DATA" result. In these cases, you may want to adjust the timeout value, perhaps first trying the maximum (ie use AT ST FF). Many vehicles will simply not support these extra addressing modes.

The other, and more common method of obtaining information is by physical addressing, in which you direct your request to a specific device, not to a functional group. To do this, you again need to construct a set of header bytes that direct your query to the physical address of the processor, or ECU. If you do not know the address, recall that the sender of information is usually shown in the third byte of the header. By monitoring your system for a time with the headers turned on (AT H1), you can quickly learn the main addresses of the senders. The SAE document J2178 assigns address ranges to these devices if you are unsure of which might be most appropriate.

When you know the address that you wish to 'speak to,' simply use it for the second byte in the header (assume an address of 10 for this example). Combine that with your knowledge of SAE J2178 to choose a priority/type byte (assume a value of E4 for this example, as if the vehicle is J1850 PWM). Finally, you need to identify yourself to the target, so that responses can be returned to you. As is customary for diagnostic tools, we'll use an address of F1. As before, these three bytes are then assigned to the header with the set header command:

```
>AT SH E4 10 F1
OK
```

From this point on, all messages that the ELM327 sends will use these three bytes for the header. All that needs to be done now is to request data from the vehicle. For physical addressing, this is often done using mode 22:

```
>22 11 6B
62 11 6B 00 00
```

The response to this command is of the same format to those seen for 'standard' OBD requests. The request has been repeated (with 40 added to the mode value in order to show that this is a response), and this is followed by the actual data (0000 in this case). The PIDs used with mode 22 are usually proprietary to each manufacturer and are generally not published widely, so you may have difficulty in determining the ones to use with your vehicle. Elm

Setting the Headers (continued)

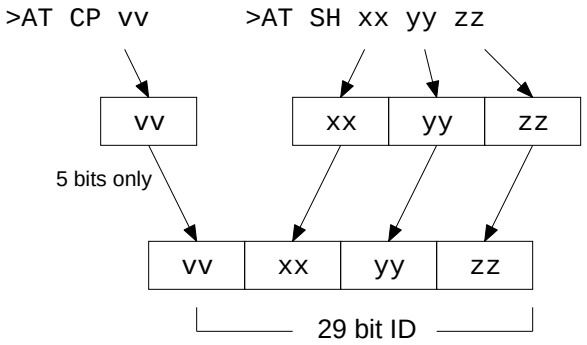


Figure 5. Setting a 29 bit (extended) CAN ID

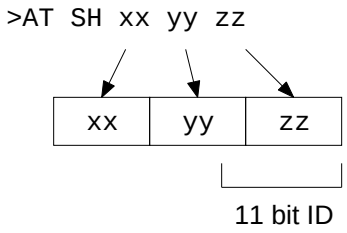


Figure 6. Setting an 11 bit (standard) CAN ID

Electronics does not maintain lists of this information, and cannot provide any further details for you. Mode 22 and others are described in more detail in SAE document J2190, "Enhanced E/E Diagnostic Test Modes".

The ISO14230-4 standard defines its' header bytes a little differently. Advanced experimenters will be aware that for ISO 14230-4, the first header byte must always include the length of the data field, which varies from message to message. From that, one might assume that the you would need to redefine the header for every message that is to be sent – not so! The ELM327 always determines the number of bytes that you are sending, and inserts that length for you, in the proper place for the header that you are using. If you are using the standard ISO 14230-4 header, the length will be put into the first header byte, and you need only provide the two (most significant) bits of this byte when defining the header. What you place in the rest of the byte will be ignored by the ELM327 unless you set it to 0. If it is 0, it is assumed that you are experimenting with KWP four byte headers, and the ELM327 then creates the fourth header byte for you. Again, you do not need to provide any length to be put into this byte – it is done for you.

Addressing within the CAN (ISO 15765-4) protocols is quite similar in many ways. First, consider the 29 bit standard. The ELM327 splits the 29 bits into a CAN Priority byte and the three header bytes that we are now familiar with. Figure 5 shows how these are combined for use by the ELM327.

The CAN standard states that for diagnostics, the priority byte ('vv' in the diagram) will always be 18 (this is the default value used by the ELM327). Since it is rarely changed, it is assigned separately from the other header bytes, using the CP command. Changing of this value is usually only required if experimenting with J1939 systems.

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, which is very similar to ISO 14230-4.

Those that are familiar with the SAE J1939 standard will likely find this header structure to be very familiar (J1939 is a CAN standard for use by 'heavy-duty vehicles' such as trucks and buses). It uses slightly different terminology, but there is a direct parallel between the bytes used by J1939 for the headers and the grouping of the bytes in the ELM327. Refer to page 41 for a discussion of the J1939 support that is provided by the ELM327.

The final header format to discuss is that 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 ELM327 uses the same commands to set these values as for other headers, except that it only uses the 11 least significant ('right-most') bits of the provided header bytes, and ignores the others (as shown in Figure 6). It quickly becomes inconvenient to have to enter six digits when only three are required, so there is a special 'short' version of the

Setting the Headers (continued)

AT SH command that uses only three hex digits. It actually operates by simply adding the leading 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:

```
>01 00
7E8 06 41 00 BE 3F B8 13 00
```

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

```
>AT SH 7E0
OK

>01 05
7E8 03 41 05 46 00 00 00 00
```

Hopefully this has helped to get you started. As we often tell those that write – if you are planning to do some serious experimenting with OBD, you should buy the relevant standards.

Monitoring the Bus

Some vehicles use the OBD bus for information transfer during normal vehicle operation, passing a great deal of information over it. A lot can be learned if you have the good fortune to connect to one of these vehicles, and are able to decipher the contents of the messages.

To see how your vehicle uses the OBD bus, you can enter the ELM327's 'Monitor All' mode, by sending the command AT MA from your terminal program. This will cause the IC to display any information that it sees on the OBD bus, regardless of transmitter or receiver addresses (it will show all). Note that the ELM327 remains silent while monitoring, so periodic 'wakeup' messages are not sent (if you have an ISO 9141 or ISO 14230 bus that had been initialized previously, it may 'go to sleep'), IFRs are not sent, and the CAN module does not acknowledge messages.

The monitoring mode can be stopped by putting a logic low level on the RTS pin, or by sending a single RS232 character to the ELM327. Any convenient character can be used to interrupt the IC – there are no restrictions on whether it is printable, etc. Note that the character you send will be discarded, and will have no effect on any subsequent commands. The time it takes to respond to this interrupting character will depend on what the ELM327 is doing when the character is received. The IC will always finish a task that is in progress (printing a line, for example) before returning to wait for input, so you should always wait for the prompt character ('>'), or the Busy line to go low, before beginning to send a command.

One unexpected result may occur if you have the automatic protocol search feature enabled, and you tell the ELM327 to begin monitoring. If the bus is quiet, the ELM327 will begin searching for an active protocol, which may not be what you were expecting. Be aware also that the ISO 9141 and ISO 14230 protocols look identical when monitoring, so the ELM327 will likely stop searching at ISO 9141, even if the actual protocol is ISO 14230. With the Automatic searching enabled, this should correct itself, however, when an OBD request is later made.

If the 'Monitor All' command provides too much information (it certainly does for most CAN systems!), then you can restrict the range of data that is to be displayed. Perhaps you only want to see messages that are being transmitted by the ECU with address 10. To do that, simply type:

```
>AT MT 10
```

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

Using this command with 11 bit CAN systems can be a little confusing at first. Recall the way in which all header bytes are stored within the ELM327. An 11 bit 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 all that look like 'x10').

The other monitoring command that is very useful is the AT MR command, which looks for specific addresses in the middle byte of the header. Using this command, you can look for all messages being sent to a particular address. For example, to use it to look for messages being sent to the ECU with address 10, simply send:

```
>AT MR 10
```

and all messages that contain 10 in the second byte of the header will be displayed.

Using this command with the 11 bit CAN systems will again need further explanation. It may be helpful to first picture the hex number '10' in the above example as the binary number '0001 0000'. Recall from above that 11 bit CAN IDs are actually stored as the least significant 11 bits in the 3 byte 'header storage' locations, and only 3 bits are actually stored in the middle byte (receiver's address) position. When comparing the received CAN ID to the address you provide with the MR command then, only the right-most 3 bits of your MR address are considered and the other 5 bits are ignored. In this example, the AT MR 10 effectively becomes AT MR 0 for 11 bit CAN systems, and all messages that begin with '0' as the first digit will actually be displayed.

In order to use the AT MR command with CAN 11 bit identifiers, you should always try to use the format 'AT MR 0x', where 'x' is the digit that you want the identifiers to begin with. To look for all 2xx's, use the command 'AT MR 02', and to see all of the 7xx's, you should use 'AT MR 07'.

The ELM327 can be set to automatically send the Monitor All command to itself after power on, if PP 00 is set to the value 0, and enabled.

CAN Messages and Filtering

The ELM327 monitoring commands allow for some filtering of the information (with AT MR and MT), but those commands still only allow one byte to be defined, of all the header bits. With CAN systems, the large volume of information present may mean that this is not precise enough, and there needs to be a better means of being more selective.

The ELM327's internal CAN module has a 'filter' that can be used to select all of the ID bits, not just one byte of them. The easiest way to use this filter is with the CRA command. For example, if the only messages that you wish to see are those that have the CAN ID 7E9, then simply send:

```
>AT CRA 7E9
```

and the ELM327 will set internal values so that all that is displayed are messages with ID 7E9.

If you do not want to define all 11 or 29 bits, a range of values can also be defined, but that requires also setting a 'mask', in addition to the filter.

As an example, consider an application where you are trying to monitor for 29 bit CAN diagnostic messages, exactly like the ELM327 does. By definition, these messages will be sent to the scan tool at address F1, so from ISO 15765-4, you know 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, 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 ELM327 to ignore those last two 0's? You do that with the mask. The mask is a bit pattern that tells the ELM327 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 this mask then, you would need to use the CAN Mask command, as follows:

```
>AT CM 1F FF FF 00
```

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 ELM327 offers some shorter versions of the CF and CM commands. To use them for the example above, you need only enter three digit arguments:

```
>AT CF 600
```

and

```
>AT CM F00
```

As for the 29 bit versions, only the 11 least significant (right-most) digits are actually 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 ELM327 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 all have the extra benefit that while they are in effect, the ELM327 will remain quiet, not sending acknowledgement or error signals, so anything you do while monitoring should not disrupt other devices that are on the bus.

Note that if a filter has been set, it will be used for all CAN messages, so setting filters and masks may cause standard OBD requests to be ignored, and you may begin seeing "NO DATA" responses. If this happens, and you are unsure of why, you may want to reset everything to their default values (with AT D) and start over.

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 serial number of the vehicle (mode 09, PID 02). This is often a multiline reply that needs to be joined. In these situations, you must take care to ensure that all of the reply has been received and it is in the correct order before assembling the message. The actual response usually has a byte that shows the sequence of the data, to help with this. Here is one example for a typical SAE J1850 vehicle:

```
>0902
49 02 01 00 00 00 31
49 02 02 44 34 47 50
49 02 03 30 30 52 35
49 02 04 35 42 31 32
49 02 05 33 34 35 36
```

Note that all OBD compliant vehicles do not necessarily provide this information. Many older ones do not, but as a rule the newer ones do. 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 an 09 02 request. The next byte on each line shows the order in which the data is to be assembled. Assembling the remainder of the data in that order, and ignoring the first few 00's gives:

```
31 44 34 47 50 30 30 52 35 35 42 31 32
33 34 35 36
```

Using an ASCII table to convert these hex digits gives the following serial number for the vehicle:

```
1 D 4 G P 0 0 R 5 5 B 1 2 3 4 5 6
```

CAN systems will display this information in a somewhat different fashion. Here is a typical response from a CAN vehicle:

```
>0902
014
0: 49 02 01 31 44 34
1: 47 50 30 30 52 35 35
2: 42 31 32 33 34 35 36
```

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 an 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 (the '01'), which shows the number of data items that are to follow (the vehicle can only have one VIN, so the response says there is only one data item). That third byte can be ignored, so this leaves 17 data bytes which are the serial number (purposely chosen to be identical to the those of the previous example). All that is needed is a conversion to ASCII, in order to read them, exactly as before.

The following shows an example of a different type of multiline response that can occur when two or more ECUs respond to one request. Here is a typical response to an 01 00 request:

```
>01 00
41 00 BE 3E B8 11
41 00 80 10 80 00
```

This is difficult to decipher without knowing a little more information. We need to turn the headers on to actually see 'who' is doing the talking:

```
>AT H1
OK

>01 00
48 6B 10 41 00 BE 3E B8 11 FA
48 6B 18 41 00 80 10 80 00 C0
```

Now, if you analyze the header, you can see that the third byte shows ECU 10 (the engine controller) and ECU 18 (the transmission) are both responding with a reply that is valid for them. This type of response occurs often, and you should be prepared for it.

A final example shows how similar messages

Multiline Responses (continued)

might occasionally be 'mixed up' in a CAN system. We ask the vehicle for the Calibration ID (09 04) and are presented with the following response:

```
>09 04
013
0: 49 04 01 35 36 30
1: 32 38 39 34 39 41 43
013
0: 49 04 01 35 36 30
2: 00 00 00 00 00 00 31
1: 32 38 39 35 34 41 43
2: 00 00 00 00 00 00 00
```

which is quite confusing. The first group (the 013, 0:, 1: group) seems to make some sense, but the following group is very confusing. Why are there two segment twos? Which ECU do these belong to? The only way to tell is to turn on the headers, and repeat your request:

```
>AT H1
OK

>09 04
7E8 10 13 49 04 01 35 36 30
7E8 21 32 38 39 34 39 41 43
7E9 10 13 49 04 01 35 36 30
7E8 22 00 00 00 00 00 00 31
7E9 21 32 38 39 35 34 41 43
7E9 22 00 00 00 00 00 00 00
```

This time, the order appears to be the same, but be aware that it may not be – that is why the standard requires that sequence codes be transmitted with multiline responses.

Looking at the first digits of these responses, you can see that some begin with 7E8, and some begin with 7E9. This is the special CAN IDs representing ECU#1 and ECU#2, respectively. Grouping the responses from the two ECUs gives:

```
7E8 10 13 49 04 01 35 36 30
7E8 21 32 38 39 34 39 41 43
7E8 22 00 00 00 00 00 00 31
```

and

```
7E9 10 13 49 04 01 35 36 30
7E9 21 32 38 39 35 34 41 43
7E9 22 00 00 00 00 00 00 00
```

From these, the messages can be assembled in their proper order. To do this, look at the byte following the CAN ID - it is what is known as the PCI byte, and is used to tell what type of data follows. In this case, the PCI byte begins with either a 1 (for a 'First Frame' message), or a 2 (for the 'Consecutive Frames'). The second half of the PCI byte shows the order in which the information is to be assembled (ie. the segment number). In this case, the segment numbers are already in order, but if they had not been, it would have been necessary to rearrange the messages to place them in order.

Each OBD standard has some minor peculiarities. Hopefully this has helped you with some of the more difficult ones. If you are still having trouble, we urge you to purchase the relevant standard, and study it.

CAN Message Formats

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

SF - the Single Frame

FF - the First Frame (of a multiframe message)

CF - the Consecutive Frame (“ “)

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. If the CAN Auto Formatting option is on (CAF1) then the ELM327 will create this byte for you when sending, and remove it for you when receiving. (If the headers are enabled, you will always see it.)

If you turn the Auto Formatting off (with CAF0), it is expected that you will provide all of the data bytes to be sent. For diagnostics systems, this means the PCI byte and the data bytes. The ELM327 will not modify your data in any way, except to add extra padding bytes for you, to ensure that you always send as many data data bytes as are required (eight for ISO15765). You do not need to set the Allow Long (AT AL) option in order to send eight bytes, as the IC overrides it for you.

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 descripter 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 determine the order when 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, so the single digit is normally more than adequate. As seen previously, the serial number for a vehicle is often a multiframe response:

```
>0902
014
0: 49 02 01 31 44 34
1: 47 50 30 30 52 35 35
2: 42 31 32 33 34 35 36
```

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 ELM327 and printed on the first line as shown. Following the First Frame line are two Consecutive Frames (that begin with 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 ELM327 must tell the sender some technical things (such as how long to delay between Consecutive Frames, etc.) and does so by replying immediately with a Flow Control message. These are predefined by the ISO 15765-4 standard, so can be automatically inserted for you. If you wish to generate custom Flow Control messages, then refer to the next section.

If a Flow Control frame is detected while monitoring, the line will be displayed with ‘FC: ’ before the data, to help you with decoding of the information.

There is a final type of message that is occasionally reported, but is not supported by the diagnostics standard. The (Bosch) CAN standard allows for the transmission of a data request without sending any data in the requesting message. To ensure that the message is seen as such, the sender also sets a special flag in the message (the RTR bit), which is seen at each receiver. The ELM327 always looks for this flag, or for zero data bytes, and may report to you that an RTR was detected while monitoring. This is shown by the characters RTR where data would normally appear, but only if the CAN Auto Formatting is off, or headers are enabled. Often, when monitoring a CAN system with an incorrect baud rate chosen, RTRs may be seen.

Note that the CAN system is quite robust with several error detecting methods in place, so that during normal data transmission you will rarely see any errors. When monitoring buses however, you may well see errors (especially if the ELM327 is set to an incorrect baud rate). As a diagnostic aid, when errors do occur, the ELM327 will print all bytes (no matter what CAF, etc., is set to), followed by the message ‘<RX ERROR’.

Altering Flow Control Messages

ISO 15765-4 (CAN) provides for only eight data bytes per frame of data. Of course, there are many cases where the data which needs to be sent is longer than 8 bytes, and CAN has made provision for this by allowing data to be separated into segments, then recombined at the receiver.

To send one of these multi-line messages, the transmitter in a CAN system will send a 'First Frame' message, and then wait for a reply from the receiver. This reply, called a 'Flow Control' message contains information concerning acceptable message timing, etc., and is required to be sent before the transmitter will send any more data. For ISO 15765-4, the type of response is well defined, and never changes. The ELM327 will automatically send this ISO 15765-4 Flow Control response for you as long as the CAN Flow Control option is enabled (CFC1), which it is by default.

Several users have requested that we provide more flexibility over the data sent in the Flow Control message, and as of v1.1, we have provided a means to do this. In order to change how the ELM327 responds when it needs to send a Flow Control message, you need to change Flow Control 'modes'.

The default Flow Control mode is number '0'. At any time while you are experimenting, if you should wish to restore the automatic Flow Control responses (for ISO 15765-4), simply set the mode to 0:

```
>AT FC SM 0
OK
```

This will immediately restore the responses to their default settings.

Mode 1 has been provided for those that need complete control over their Flow Control messages. To use it, simply define the CAN ID (header) and data bytes that you require be sent in response to a First Frame message. If you try to set the mode before defining these values, you will get an error:

```
>AT FC SM 1
?
```

You must set the headers and data first:

```
>AT FC SH 7E8
OK

>AT FC SD 00 11 22
OK
```

then you can set the mode:

```
>AT FC SM 1
OK
```

From this point on, every First Frame message received will be responded to with the custom message that you have defined (7E8 00 11 22 in this example).

The final mode currently supported allows the user to set the data bytes which are to be sent, but not the ID bits. The ID bits (header bytes) in mode 2 are set to those which were received in the First Frame message, without change. To use this mode, first define your data bytes, then activate the mode:

```
>AT FC SD 00 11 22
OK

>AT FC SM 2
OK
```

For most people, there will be little need to manipulate these 'Flow Control' messages, as the defaults are designed to work with the CAN OBD standards. If you wish to experiment, these special AT commands offer that control for you.

Figure 7 summarizes the currently supported modes:

FC Mode	ELM327 Provides	User Provides
0	ID Bits Data Bytes	no values
1	no values	ID Bits Data Bytes
2	ID Bits	Data Bytes

Figure 7. Flow Control Mode Numbers

J1939 Support

The SAE J1939 CAN standard is a relatively new protocol that is being used in many types of heavy machinery – trucks, agricultural equipment, and busses, to name a few. It uses a standard CAN (ISO 11898) physical interface, and defines its own format for data transfer (although it is very similar to the ISO 15765 standard that is used for automobiles).

The ELM327 offers some support for the J1939 standard in Protocol A. (Protocols B and C may also be selected for J1939 formatting if needed, with Programmable Parameters 2C and 2E). Although this is not full support as dictated by the standards, the ELM327 implementation should at least allow you to begin experimenting.

The default data rate for Protocol A has been set to 250 kbps, as is required by the SAE J1939-11 standard. Should your application require a different baud rate, it is easily changed with PP 2B.

J1939 messages use 29 bit CAN IDs, and up to 8 data bytes for each message (ISO 15765 always uses 8 bytes). Diagnostics on SAE J1939 are defined by the J1939-73 standard, and the actual data transfer protocol is described in J1939-21. If you are going to do a lot of work with J1939, it may be wise to purchase these standards documents from the Society of Automotive Engineers (SAE).

The ELM327 handles the sending of messages with the J1939 protocol in the same way that it does the other protocols. The header bytes have been predefined for you (to send a global request for PGN data), and all you need do is provide the PGN number as data bytes.

For example, assume that you wish to send a request for engine temperature (which is PGN 00FEEE). Simply send the request as you would any other OBD request:

```
>00 FE EE
```

When it receives these bytes, the ELM327 adds the required header (ID) bytes, etc. and sends the message for you. It then configures itself to be able to receive a response, and prints out anything that it sees.

If you wish to send a request to a specific address rather than globally, you will need to change the header bytes to that address prior to requesting the data. Setting these bytes is done in exactly the same way as with other protocols - with the AT SH command. For example, the if you wish to send a request directly to Engine #1 (address 00), from the scan tool (F9), you would simply send the command AT SH EA 00 F9 prior to sending any data. After that, all data will be directed to the engine, so don't forget to reset the headers if you wish to again make global requests. Note that the ELM327 does not negotiate for a bus address (per J1939-81), and always uses the address set by the header bytes.

If you are familiar with the standard, you will notice that J1939 actually specifies a reverse order for the data bytes (the above request should actually be sent as EE FE 00), but this can be very confusing, so the ELM327 automatically reverses the order for you. This feature can be disabled at any time with the AT JS command.

All responses to a request are printed by the ELM327, whether they are a single CAN message, or a multisegment transmission as defined by the transport protocol (J1939-21). If the responses are multisegment, the ELM327 handles all of the negotiation for you (transparently - you won't see it happen). Note that if you provide three data bytes in the request, the ELM327 will assume that you are sending a PGN, and will look for responses to that. If you send other than 3 data bytes, it will assume that you are making a general request, and it will only look for responses to the Source Address (as given in the third byte of the header).

The SAE J1939 standard predefines several Diagnostic Messages, and assigns ID values to them. The first of these (designated 'DM1') is reserved for trouble codes, which are periodically broadcast over the CAN network. Since these are commonly required, the ELM327 provides a special command to monitor for them (aptly named DM1). If you wish to monitor for all DM1 messages, simply issue:

```
>AT DM1
```

and the ELM327 will print out all that it finds.

The DM1 message is the only one that is presently predefined by the ELM327 for you. To monitor for any other diagnostic message, you need to know it's PGN number (in hexadecimal), and monitor for that using the AT MP command. For example, the DM2 PGN number is 65227, or 00FECB in hexadecimal, so to monitor for DM2s, you need only send:

```
>AT MP 00FECB
```

and the IC will begin looking for all replies to DM2 requests. Since PGNs presently all begin with 00, you

J1939 Support (continued)

may find it easier to simply type a shorter version of this command:

```
>AT MP FECB
```

and have the ELM327 automatically add the two zeros for you. This shorter version of the command is presented as a convenience, and is equivalent to the other version.

Received messages are presented in a slightly different way with the J1939 protocols, when the automatic formating option is selected (it is by default). As an example, we will reset the IC, then set the protocol to J1939, and then turn the display of headers on. The commands required to perform these steps are as follows:

```
>AT Z


ELM327 v1.3a

>AT SP A
OK

>AT DP
SAE J1939 (CAN 29/250)

>AT H1
OK
```

now, turn the auto formatting off, and look at what may typically be seen:

```
>AT CAF0
OK

>AT DM1
18 FE CA 00 40 FF B8 04 03 8A FF FF
18 FE CA 00 40 FF B8 04 03 8A FF FF
18 FE CA 00 40 FF B8 04 03 8A FF FF
```

You will need to stop this monitoring mode by pressing any key. When the prompt character appears again, try the same command, but with formatting on:

```
>AT CAF1
OK

>AT DM1
6 0FECA 00 40 FF B8 04 03 8A FF FF
6 0FECA 00 40 FF B8 04 03 8A FF FF
6 0FECA 00 40 FF B8 04 03 8A FF FF
```

Again, the monitoring will need to be stopped at some point, by pressing any key.

If you are familiar with the SAE J1939 standard, you will likely recognize how the ELM327 formats the J1939 data. The 6 at the beginning represents the priority value, which is a 3 bit value set by the standard. The 0FECA represents the identifier value assigned to the DM1 PGN (the 0 before the FECA is the Data Page bit). Note that the ELM327 does not print the Reserved bit value when formatting is on - if you need to see that, you will have to turn the formatting off.

Immediately following the 0FECA is the value 00, which represents the Source Address. In this case, it is 00 for Engine #1. If the data length code had been turned on (with AT D1), a single digit would have appeared next, but since we have it off, there is nothing. The final 8 bytes represent the data bytes as transmitted by the message (these were not taken from a vehicle, and are only shown as an example). There are 8 bytes here, but data lengths are defined for each PGN, and while they are often 8 bytes long, they are not required to be.

This has been a brief description of the ELM327's support for the SAE J1939 standard. Please note that it is only basic (it does not support address negotiation as described in J1939-81, for example), and it has not undergone the extensive testing that our automotive protocols have. We simply have had a large number of requests for support of the protocol, and wanted to provide something for you to begin experimenting with. As always, we are quite open to your feedback and your suggestions for future upgrades.

Restoring Order

There may be times when it seems the ELM327 is out of control, and you will need to know how to restore order. Before we continue to discuss modifying too many parameters, this seems to be a good point to discuss how to 'get back to the start'. Perhaps you have told the ELM327 to monitor all data, and there are screens and screens of data flying by. Perhaps the IC is now responding with 'NO DATA' when it did work previously. This is when a few tips may help.

The ELM327 can always be interrupted from a task by a single keystroke from the keyboard. As part of its normal operation, checks are made for received characters, and if found the IC will stop what it is doing at the next opportunity. Often this means that it will continue to send the information on the current line, then stop, print a prompt character, and wait for your input. The stopping may not always seem immediate if the RS232 send buffer is almost full, though – you will not actually see the prompt character until the buffer has emptied, and your terminal program has finished printing what it has received.

There are times when the problems seem more serious and you don't remember just what you did to make them so bad. Perhaps you have 'adjusted' some of the timers, then experimented with the CAN filter, or perhaps tried to see what happens if the header bytes were changed. All of these can be reset by sending the 'set to Defaults' AT Command:

```
>AT D
OK
```

This will often be sufficient to restore order, but it can occasionally bring unexpected results. One such surprise will occur if you are connected to a vehicle using one protocol, but the saved (default) protocol is a different one. In this case, the ELM327 will close the current session and then change the protocol to the default one, exactly as instructed.

If the AT D does not bring the expected results, it may be necessary to do something more drastic - like resetting the entire IC. There are a few ways that this can be performed with the ELM327. One way is to simply remove the power and then reapply it. Another way that acts exactly the same way as a power off and then on is to send the full reset command:

```
>AT Z
```

It takes approximately one second for the IC to perform this reset, initialize everything and then test the four status LEDs in sequence. A much quicker option is available with the ELM327, however, if the led test is not required – the 'Warm Start' command:

```
>AT WS
```

The AT WS command performs a software reset, restoring exactly the same items as the AT Z does, but it omits the LED test, making it considerably faster. Also, it does not affect any baud rates that have been set with the AT BRD command, so is essential if you are modifying the RS232 baud rates with software.

Any of the above methods should be effective in restoring order while experimenting. There is always the chance that you may have changed a Programmable Parameter, however, and are still having problems with your system. In this case, you may want to simply turn off all Programmable Parameters (which forces them to their default values). To do so, send the command:

```
>AT PP FF OFF
```

which should disable all of the changes that you have made. You can then start over with what is essentially a device with 'factory settings'. There may be times when even this command is not recognized, however. If that is the case, you will need to use the hardware method of turning the PPs off (see the next section on 'Programmable Parameters', for more details).

Programmable Parameters

The ELM327 contains several programmable memory locations that retain their data even after power is turned off. Every time the IC is powered up, these locations are read and used to change the default settings for such things as whether to display the headers, or how often to send 'wakeup' messages.

The settings, or parameters, can be altered by the user at any time using a few simple commands. These Programmable Parameter commands are standard AT Commands, with one exception: each one requires a two-step process to complete. This extra step provides some security against random inputs that might try to make changes.

The following pages list the currently supported Programmable Parameters for this version of the ELM327. As an example of how to use a Programmable Parameter, consider PP 01 (the printing of headers). If you are constantly powering your ELM327 and then using AT H1 to turn the headers on, you may want to change its default setting, so that they are always on by default. To do this, simply set the value of PP 01 to 00:

```
>AT PP 01 SV 00
OK
```

This changes the value associated with PP 01, but does not enabled it. To make the change effective, you must also type:

```
>AT PP 01 ON
OK
```

At this point, you have changed the default setting for AT H1/H0, but you have not changed the actual value of the current AT H1/H0 setting. From the 'Type' column in the table on page 45, you can see that the change only becomes effective the next time that defaults are restored. This could be from a reset, a power off/on, or possibly an AT D command.

With time, it may be difficult to know what changes you have made to the Programmable Parameters. To help with that, the ELM327 provides a Programmable Parameter Summary (PPS) command. This simply prints a list of all of the supported PPs, their current value, and whether they are on/enabled (N), or off/disabled (F). For an ELM327 v1.3a IC, with only the headers enabled (as discussed above), the summary table would look as follows:

```
>AT PPS
00:FF F  01:00 N  02:FF F  03:32 F
04:01 F  05:FF F  06:F1 F  07:09 F
08:FF F  09:00 F  0A:0A F  0B:FF F
0C:68 F  0D:0D F  0E:FF F  0F:FF F
10:0D F  11:00 F  12:FF F  13:32 F
14:FF F  15:FF F  16:FF F  17:92 F
18:00 F  19:FF F  1A:FF F  1B:FF F
1C:FF F  1D:FF F  1E:FF F  1F:FF F
20:FF F  21:FF F  22:FF F  23:FF F
24:00 F  25:00 F  26:00 F  27:FF F
28:FF F  29:FF F  2A:38 F  2B:02 F
2C:E0 F  2D:04 F  2E:80 F  2F:0A F
```

You can see that PP 01 now shows a value of 00, and it is enabled (oN).

Another example shows how you might change the CAN filler byte. Some systems use 'AA' as the value to put into unused CAN bytes, while the ELM327 uses '00' by default. To change the ELM327's behaviour, simply change PP 26:

```
>AT PP 26 SV AA
OK

>AT PP 26 ON
OK
```

Again, PP 26 is of type 'D', so the above change will not actually take effect until the AT D command is issued, or the ELM327 is reset.

The Programmable Parameters are a great way to customize your ELM327 for your own use, but you should do so with caution if using commercial software. Most software expects an ELM327 to respond in certain ways to commands, and may be confused if the headers are on when not expected, or if the CAN response shows data length codes, for example. If you make changes, it might be best to make small changes and then see the effect of each, so that it is easier to retrace your steps and 'undo' what you have done. If you get in too deeply, don't forget the 'all off' command:

```
>AT PP FF OFF
```

No matter what software you use, you might get into more serious trouble, should you change the baud rate, or the Carriage Return character, for example, and forget what you have set them to. The Carriage Return value set by PP 0D is the only character that is recognized by the ELM327 as ending a command, so

Programmable Parameters (continued)

if you change its value, you may not be able to undo your change. In this case, your only recourse may be to force all of the PPs off with a hardware trick.

When the ELM327 first powers up, it looks for a jumper between pin 28 (the OBD Tx LED output) and circuit common (V_{SS}). If a jumper is in place, it will turn off all of the PPs for you, restoring the IC to the factory defaults. To use this feature, simply connect a jumper to circuit common (which appears in numerous places - pins 8 or 19 of the ELM327, pin 5 of the RS232 connector, one end of most capacitors, or at the OBD connector), then hold the other end of the jumper to pin 28 while turning the power on. When you see the RS232 Rx LED begin to flash quickly, remove the jumper – the PPs are off.

This feature should only be used when you get into trouble too deeply, and it's your only choice (since putting jumpers into a live circuit might cause damage if you put it into the wrong place). As well, it is only available beginning with version 1.2 of the IC, and can not be used with any previous versions.

The following pages provide a summary of the currently available Programmable Parameters. Note that the "Type" column indicates when changes to the Programmable Parameter will take effect. Possible values are:

I - the effect is Immediate,

D - takes effect after Defaults are restored
(AT D, AT Z, AT WS, MCLR or power off/on)

R - takes effect after any Reset
(AT Z, AT WS, MCLR or power off/on)

P - needs a Power off/on type reset
(AT Z, MCLR, or power off/on)

Programmable Parameter Summary

PP	Description	Values	Default	Type
00	Perform AT MA after powerup or reset	00 = ON FF = OFF	FF (OFF)	R
01	Printing of header bytes (AT H default setting)	00 = ON FF = OFF	FF (OFF)	D
02	Allow long messages (AT AL default setting)	00 = ON FF = OFF	FF (OFF)	D
03	NO DATA timeout time (AT ST default setting) setting = value x 4.096 msec	00 to FF	32 (205 msec)	D
04	Default Adaptive Timing mode (AT AT setting)	00 to 02	01	D
06	OBD Source (Tester) Address. Not used for J1939 protocols.	00 to FF	F1	R
07	Last Protocol to try during automatic searches	01 to 0C	09	I
09	Character echo (AT E default setting)	00 = ON FF = OFF	00 (ON)	R
0A	Linefeed Character	00 to FF	0A	R

Programmable Parameter Summary (continued)

<table>
<tr><th>PP</th><th>Description</th><th>Values</th><th>Default</th><th>Type</th></tr>
<tr><td>0C</td><td>RS232 baud rate divisor when pin 6=1
The baud rate (in kbps) is given by 4000/(this value)
For example, to obtain 500kbps, use a setting of 08 (4000/8 = 500)

Here are some example baud rates, and the divisor to be used:

<table>
<tr><th>Baud Rate (kbps)</th><th>PP 0C value hex (dec)</th></tr>
<tr><td>19.2</td><td>D0 (208)</td></tr>
<tr><td>38.4</td><td>68 (104)</td></tr>
<tr><td>57.6</td><td>45 (69)</td></tr>
<tr><td>115.2</td><td>23 (35)</td></tr>
<tr><td>230.4</td><td>11 (17)</td></tr>
<tr><td>500</td><td>08 (8)</td></tr>
</table>

Note that the PP 0C value must be provided as hex digits only.
The decimal values (in brackets) are shown for your convenience.</td><td>08 to FF</td><td>68
(38.4)</td><td>P</td></tr>
<tr><td>0D</td><td>Carriage Return Character</td><td>00 to FF</td><td>0D</td><td>R</td></tr>
<tr><td>10</td><td>J1850 voltage settling time
setting = value x 4.096 msec</td><td>00 to FF</td><td>0D
(53 msec)</td><td>I</td></tr>
<tr><td>11</td><td>J1850 Break Signal monitor enable
(reports BUS ERROR if break signal duration limits are exceeded)</td><td>00 = ON
FF = OFF</td><td>00
(ON)</td><td>D</td></tr>
<tr><td>13</td><td>Auto search time delay between protocols 1 & 2
setting = value x 4.096 msec</td><td>00 to FF</td><td>32
(205 msec)</td><td>I</td></tr>
<tr><td>16</td><td>Default ISO baud rate (AT IB default setting)</td><td>00 = 96
FF = 10</td><td>FF
(10.4K)</td><td>R</td></tr>
<tr><td>17</td><td>ISO wakeup message rate (AT SW default setting)
setting = value x 20.48 msec</td><td>00 to FF</td><td>92
(2.99 sec)</td><td>D</td></tr>
<tr><td>18</td><td>Auto search time delay between protocols 4 & 5
setting = value x 4.096 msec</td><td>00 to FF</td><td>00
(no delay)</td><td>I</td></tr>
<tr><td>24</td><td>CAN auto formatting (AT CAF default setting)</td><td>00 = ON
FF = OFF</td><td>00
(ON)</td><td>D</td></tr>
<tr><td>25</td><td>CAN auto flow control (AT CFC default setting)</td><td>00 = ON
FF = OFF</td><td>00
(ON)</td><td>D</td></tr>
<tr><td>26</td><td>CAN filler byte (used to pad out messages)</td><td>00 to FF</td><td>00</td><td>D</td></tr>
</table>

Programmable Parameter Summary (continued)

PP	Description	Values	Default	Type
29	Printing of the CAN data length (DLC) when printing header bytes (AT D0/D1 default setting)	00 = ON FF = OFF	FF (OFF)	D
2A	CAN Error Checking (applies to protocols 6 to C) Each bit of this byte controls an option, as follows: b7: ISO15765 Data Length 0=accept any 1=must be 8 bytes b6: ISO15765 PCI=00 0=allowed 1=not allowed b5: Search after ERR94 0=normal 1=CAN is blocked b4: Search after LV RESET 0=normal 1=CAN is blocked b3: Wiring Test 0=bypass 1=perform b2 to b0: reserved for future - leave set to 0 Note that b7 is the msb, and b0 is the lsb.	00 to FF	38 (00111000)	D
2B	Protocol A (SAE J1939) CAN baud rate divisor. The protocol A CAN baud rate is determined by this value (rate in kbps = 500/value). For example, setting this PP to 19 (ie. decimal 25) provides a baud rate of 500/25 = 20 kbps.	01 to 40	02 (250 Kbps)	R
2C	Protocol B (USER1) CAN options. Each bit of this byte controls an option, as follows: b7: Transmit ID Length 0=29 bit ID 1=11 bit ID b6: Data Length 0=fixed 8 byte 1=Variable DLC b5: Receive ID Length 0=as set by b7 1=both 11 and 29 bit b4: reserved for future - leave set at 0 b3: reserved for future - leave set at 0. b2, b1, and b0 determine the data formatting options, as follows: b2 b1 b0 Data Format 0 0 0 none 0 0 1 ISO 15765-4 0 1 0 SAE J1939 Other combinations are reserved for future – results might be unpredictable if you should select one of them. Note that b7 is the msb, and b0 is the lsb. For example, the default value of E0 can be shown as 11100000 in binary, which has b7, b6 and b5 set to 1's. This means that by default, 11 bit IDs will be sent, the data length will vary, and the ELM327 will receive messages with either 11 or 29 bit IDs.	00 to FF	E0 (11100000)	R
2D	Protocol B (USER1) baud rate divisor. See PP 2B for a description.	01 to 40	04 (125 Kbps)	R
2E	Protocol C (USER2) CAN options. See PP 2C for a description.	00 to FF	80 (10000000)	R
2F	Protocol C (USER2) baud rate divisor. See PP 2B for a description.	01 to 40	0A (50 Kbps)	R

Using Higher RS232 Baud Rates

The RS232 serial interface has been maintained throughout the ELM OBD products, largely due to its versatility. Older computers and PDAs can use it directly, as can microprocessors. Newer computers generally do not have physical RS232 ports, but they do have USB and ethernet ports, both of which can be converted to RS232 with a simple adapter.

Most people will build their ELM327 circuits with an RS232 interface, mainly because it is relatively easy and inexpensive to do. A circuit such as the one shown in Figure 9 requires very few components and works extremely well at speeds of up to 57600 bps. Depending on your computer's RS232 voltages, your wiring practices, and your choice of components, it may also work well at speeds as high as 115200 bps. This is the approximate limit of such a circuit however, and anything operating at this speed should be thoroughly tested.

Users that would like to operate at speeds in the range of 115200 bps (or higher) may wish to look at some of the single IC solutions that are available. These include devices such as the ADM232A from Analog Devices (http://www.analog.com/), or the popular MAX232 series of ICs from Maxim Integrated Products (http://www.maxim-ic.com/). These are all excellent devices that can be used for speeds of up to 115.2 Kbps. We do caution that many of these types of devices are only rated for operation up to 120 Kbps, however, so may not be suitable for higher data rates - be sure to check the manufacturers data sheet before committing to a design.

An RS232 interface needs relatively large voltage swings, which are difficult to maintain at higher rates with large cable capacitances to contend with. (A typical interface is often limited to about 230.4Kbps under ideal conditions.) If you need to operate the ELM327 at these speeds or higher, it is recommended that you consider alternatives.

One popular alternative is a USB data connection. The USB interface is capable of very high data transfer rates, certainly much higher than the 500 Kbps limit of the ELM327. Several manufacturers offer special 'bridge' circuits that simplify connecting an RS232 device (such as the ELM327) directly to the USB bus. Examples are the CP2102 from Silicon Labs (http://www.silabs.com/) or the FT232R from Future Technology Devices (http://www.ftdichip.com/). If planning to use the higher baud rates (ie up to 500 Kbaud), these interfaces are essential.

We are often asked if it is possible to use a direct connection to a microprocessor. That is certainly an option, and one that allows a full speed connection at essentially zero cost. If you are developing such an interface, refer to page 51 for more details.

The default configuration for the ELM327 provides an RS232 data rate of either 9600 baud, or 38400 baud, depending on the voltage level at pin 6 during power up or reset. While the 9600 baud rate is not adjustable, the 38400 one is (beginning with v1.2 of the IC). There are two ways that the rate can be changed – either permanently with a Programmable Parameter, or temporarily with an AT command.

Programmable Parameter '0C' is the memory location that allows you to store a new baud rate which replaces the 38.4Kbps high speed rate. The value is stored in 'non-volatile' memory (EEPROM) that is not affected by power cycles or resets (but changing this value may affect the operation of some software packages, so be careful how you use it).

If you store a new value in PP 0C, then enable it, and if pin 6 is at a high level during the next powerup, then your stored rate will become the new default data rate. (If it has not been enabled, the rate will revert to the factory default of 38.4Kbps.) As an example, perhaps you would like to have the ELM327 use a baud rate of 57.6Kbps, rather than the factory setting of 38.4Kbps. To do this, determine the required value for PP 0C, store this value in PP 0C, and then enable the PP.

The value stored in PP 0C is actually an internal divisor that is used to determine the baud rate (it will be 4000Kbps divided by the value of PP 0C). To obtain a setting of 57.6, a baud rate divisor of 69 is required (4000/69 is approximately 57.6). Since 69 in decimal is 45 in hexadecimal, you need to tell the ELM327 to set the value of PP 0C to 45, with this command:

```
>AT PP 0C SV 45
```

then enable the new value for use:

```
>AT PP 0C ON
```

from that point (until PP 0C is turned off), the default data rate will be 57.6K, and not 38.4K. Note that the value that you write does not become effective until the next full reset (a power off/on, AT Z, or MCLR pulse).

If you are designing your own circuitry, you will know what your circuit is capable of, and can assign a value to PP 0C. Software developers will not usually

Using Higher RS232 Baud Rates (continued)

know what hardware is to be connected, however, so will not know what the limitations are. For these users, we have provided the BRD command.

This command allows a new baud rate divisor to be tested, and then accepted or rejected depending on the results of the test. See the chart at the right, which shows how the command works.

As can be seen, the software first makes a request for a new baud rate divisor, using this AT command. For example, to try the 57.6K rate that was previously discussed, the controlling PC would send:

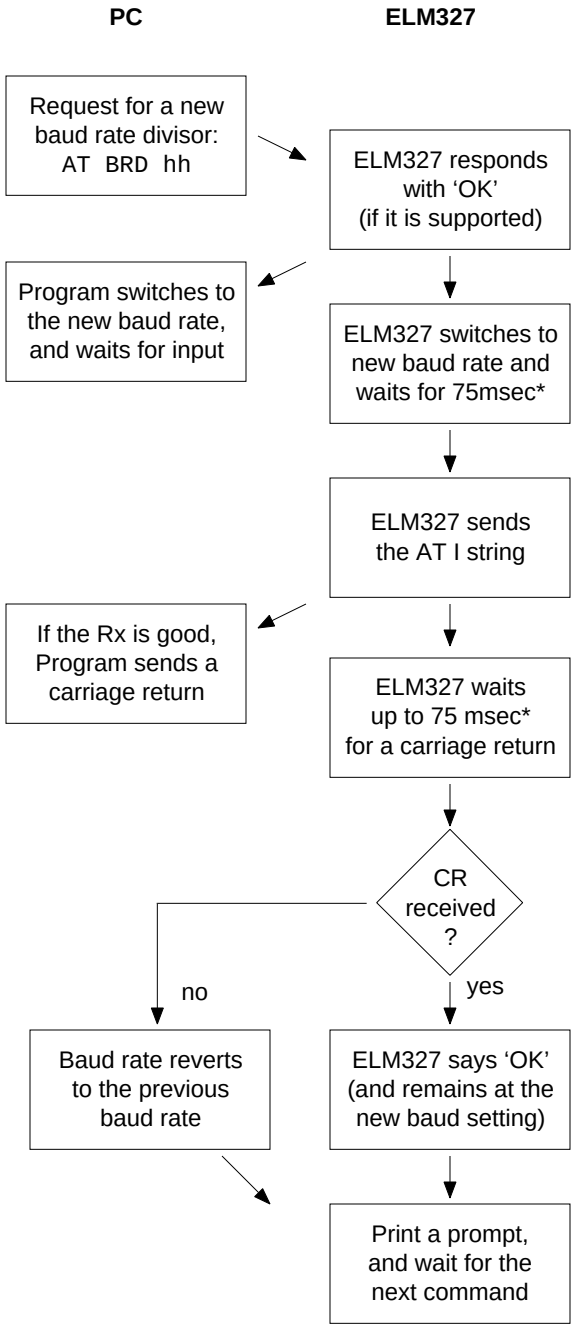
```
AT BRD 45
```

If the ELM327 is an older version, it will not support this command, and will return with the familiar '?' character. If it does support the command, it will respond with 'OK', so the software knows whether to proceed or not. No prompt character follows the 'OK' reply; it is followed only by a carriage return character (and optionally, a linefeed character).

Having sent an 'OK', the ELM327 then switches to the new (proposed) baud rate, and then simply waits a predetermined time (nominally 75 msec). This period is to allow the PC sufficient time to change its baud rate. When the time is up, the ELM327 then sends the ID string (currently 'ELM327 v1.3a') to the PC at the new baud rate (followed by a carriage return) and waits for a response.

Knowing that it should receive the ELM327 ID string, the PC software compares what was actually received to what was expected. If they match, the PC responds with a carriage return character, but if there is a problem, the PC sends nothing. The ELM327 is meanwhile waiting for a valid carriage return character to arrive. If it does (within 75 msec), the proposed baud rate is retained, and the ELM327 says 'OK' at this new rate. If it does not, the baud rate then reverts back to the old rate. Note that the PC might correctly output the carriage return at this new rate, but the interface circuitry could corrupt the character, and the ELM327 might not see a valid response, so your software must check for an 'OK' response before assuming that the new rate has been accepted.

Using this method, a program can quickly try several baud rates, and determine the most suitable one for the connected hardware. The new baud rate will stay in effect until reset by an AT Z, a Power Off/On, or a MCLR input. It is not affected by the AT D (set Defaults), or AT WS (Warm Start) commands.

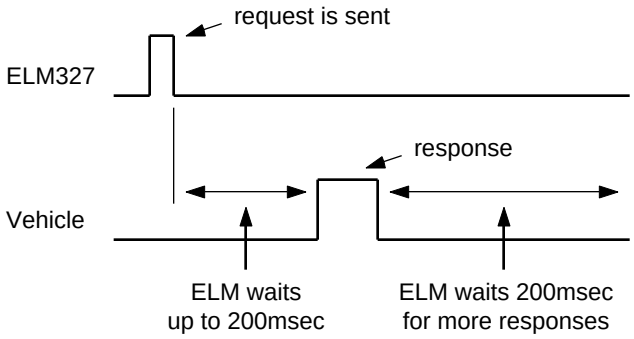


* Note that the 75msec time is adjustable (with the AT BRT hh command)

Setting Timeouts - AT ST and AT AT Commands

Users often ask about how to obtain faster OBD scanning rates. There is no definite answer for all vehicles, but the following information may help with understanding how the settings might apply to your vehicle.

A typical vehicle request and response is shown in the diagram below:



The ELM327 sends a request then waits up to 200msec for a reply. If none were to come, an internal timer would stop the waiting, and the ELM327 would print “NO DATA”.

After each reply has been received, the ELM327 must wait to see if any more replies are coming (and it uses the same internal timer to stop the waiting if no more replies arrive). With our initial OBD products (the ELM320, ELM322 and ELM323) we found that older vehicles often needed a timeout setting of about 100 msec, and occasionally they needed more, so we settled on a standard default setting of 200 msec.

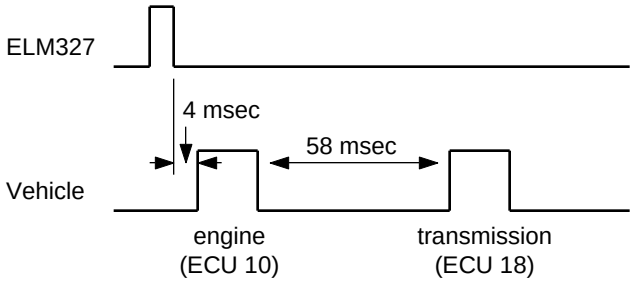
If a typical vehicle query response time were about 50 msec, and the timeout were set to 200 msec, the fastest scan rate possible would only be about 4 queries per second. Changing the ST time to about 100msec would almost double that rate, giving about 7 queries per second. Clearly, if you were to know how long it takes for your vehicle to reply, you might be able to improve on the scan rate, by adjusting the ST time.

It is not easy to tell how fast a vehicle replies to requests. For one thing, requests all have priorities assigned, so responses may be fast at some times, and slower at others. The physical measurement of the time is not easy either - it requires expensive test equipment just to make one measurement. To help with this, we added a new feature to the v1.2 IC, called Adaptive Timing.

Adaptive timing actually makes the response time measurements for you, and adjusts the AT ST time to a value that should work for most situations. It is enabled by default, but can be disabled with the AT0 command should you not agree with what it is doing (there is also an AT2 setting that is a little more agressive, should you wish to experiement). For 99% of all vehicles, we recommend that you simply leave the settings at their default values, and let the ELM327 make the adjustments for you.

Surely there has to be a way to eliminate that final timeout, if you know how many responses to expect? There is a way now, with v1.3a of the ELM327.

Consider the following times taken from a J1850 VPW vehicle, in response to an 01 00 request:



The engine responds very quickly, but the transmission takes considerably longer. The adaptive timing algorithm measures the longer transmission response times and will use them to set the timeout, likely to a value in the range of 90msec. With a timeout of 90 msec, the maximum scan rate may be about 6 readings per second.

Instead of sending 01 00 for the above request, the ELM327 will now also accept 01 00 2. This tells the IC to send 01 00, then return immediately after receiving 2 responses. It can not speed up a slow ECU, but it will eliminate the final delay, as the ELM327 now knows the number of responses to expect.

We do caution that you use this new feature carefully. If you set the last digit to a number that is less than the actual number of responses, then acknowledgements that may be required will not be sent, and some protocols may begin resending the message, looking for a response. This will lead to unnecessary network congestion, and should be avoided. Before using this feature, always determine the number of responses that will be coming from the vehicle, and then set the responses digit to that value.

Microprocessor Interfaces

A very common question that we receive is “Can I connect the ELM327 directly to my own circuit, or must I use the RS232 interface shown?” Certainly you may connect directly to our ICs, and you do not need to use an RS232 interface. There are a few items to consider, however.

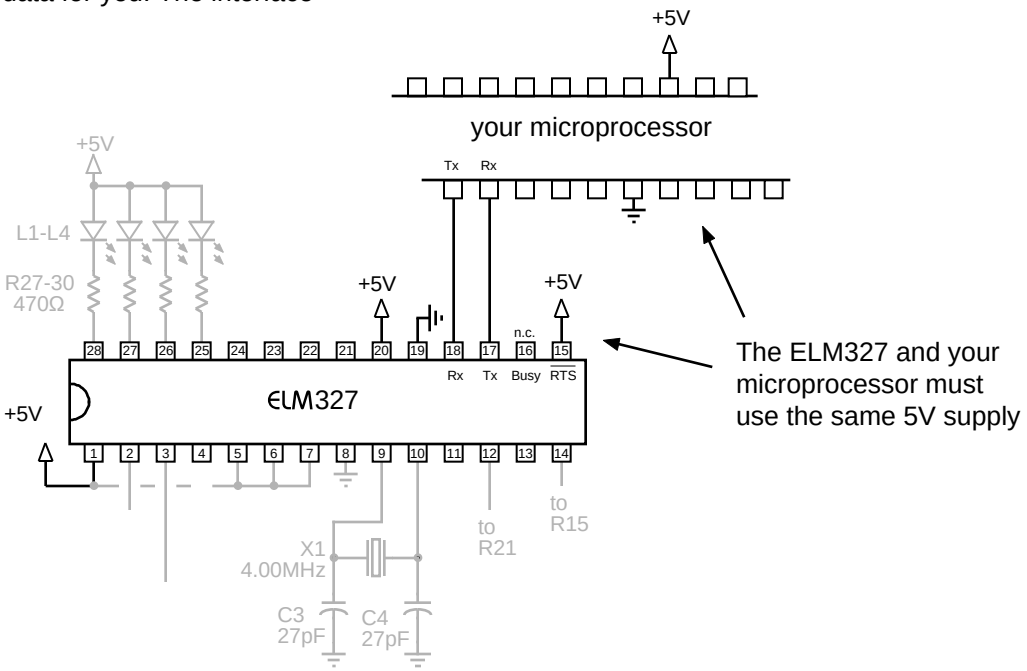
The ELM327 is actually a microprocessor that contains a standard UART type interface, connected to the RS232 Tx and Rx pins. The logic type is CMOS, and this is compatible with virtually all 5V TTL and CMOS circuits, so you should be able to connect directly to these pins provided that the two devices share the same power supply (5V), and that they are not physically more than about 10 to 20 inches apart (CMOS circuits are subject to latchup from induced currents, which may be a problem if you have long leads).

The normal (idle) levels of the ELM327 transmit and receive pins are at the V_{DD} (5V) level. Most microprocessors and RS232 interface ICs expect that to be the idle level, but you should verify it for your microprocessor before connecting to the ELM327. The connections are straightforward - transmit connects to receive, and receive connects to transmit, as shown below. Don’t forget to set both devices to the same baud rate.

The ELM327 also provides a hand-shaking feature that may simplify the flow of data for you. The interface consists of two pins - an input an an output. The input is called “request to send” (RTS), and it is used to interrupt the ELM327, just the same as tapping a key on the keyboard when using a terminal program. The output pin (“Busy”) is used by the ELM327 to tell your system that it is processing data.

To use the handshaking feature, set one of your port pins to normally provide a high output, and connect it to the RTS input (pin 15). Use another port pin as an input to monitor the ELM327 Busy output. When you want to send a command, simply check the Busy output first. If it is at a high logic level, then either wait for it to go low, or if you need to interrupt the IC, then bring the RTS line low and wait for the Busy line to go low. (You might want to consider using an edge triggered interrupt on the Busy output, if one is available). When Busy does go low, restore your RTS line to a high level, and then send your command to the ELM327. No need to worry about the ELM327 becoming busy again after you raise the RTS line at this point – once Busy goes low, the ELM327 will wait (indefinitely) for your command.

If you do not use the RTS input on the ELM327, it must be connected to a high logic level, as shown below.



Error Messages

When hardware or data problems occur, the ELM327 will respond with one of the following short messages. Some of the messages are suppressed during an automatic search for a protocol, and are only visible if not in the 'auto' mode.

Here is a brief description of each:

?

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

BUFFER FULL

The ELM327 provides a 256 byte internal RS232 transmit buffer so that OBD messages can be received quickly, stored, and sent to the computer or PDA 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 may become 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, give serious consideration to 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 ELM327 tried to send the message, or tried to initialize the bus, and detected too much activity to do so (it needs a pause in activity in order to insert the message). Although this could be because the bus was in fact very busy, it is most often due a wiring problem that is giving a continuously active input. If this is an initial trial with your ELM327 circuit, check all of the voltage levels at the offending OBD input – this error is very likely due to a wiring problem.

BUS ERROR

A generic problem has occured. This is most often from an invalid signal being detected (a pulse that is longer than a Break signal, etc.) on the bus, but could be from a wiring error. Note that some vehicles may generate long pulses as part of their startup process, so you may see this message as part of a normal vehicle startup while 'monitoring all.'

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, but it may be because you have set the system to an incorrect protocol, or to a rate that does not match the actual data rate. As with BUS ERRORs, the CAN ERROR might also be the result of a wiring problem, so if this is the first time using your ELM327 circuit, review all of your CAN interface circuitry before proceeding.

DATA ERROR

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

<DATA ERROR

There was an error in the line that this points to, either from an incorrect checksum, or a problem with the format of the message (the ELM327 still shows you what it received). There could have been a noise burst which interfered, possibly a circuit problem, or perhaps you have the CAN Auto Formatting (CAF) on and you are looking at a system that is not of the ISO 15765-4 format. Try re-sending the command again – if it was a noise burst, it may be received correctly the second time.

ERRxx

There are a number of internal errors that might be reported as ERR with a two digit code following. These occur if an internally monitored parameter is found to be out of limits, or if a module is not responding correctly. If you witness one of these, contact Elm Electronics for advice.

One error that is not necessarily a result of an internal problem is ERR94. This code represents a 'fatal CAN error', and may be seen if there are CAN network issues (some non-CAN vehicles may use pins

Error Messages (continued)

6 and 14 of the connector for other functions, and this may cause problems). If you see an ERR94, it means that the CAN module was not able to reset itself, and needed a complete IC reset to do so. You will need to restore any settings that you had previuosly made, as they will have been restored to their default values.

Beginning with v1.3a of this IC, an ERR94 will also block any automatic searches through the CAN protocols, if bit 5 of PP 2A is a '1' (it is by default). This is done because most ERR94s will be as a result of serious CAN wiring problems. Blocking of the CAN protocols remains in effect until the next power off and on, or until an AT FE is sent.

FB ERROR

When an OBD output is energized, a check is always made to ensure that the signal also appears at the respective input. If there is a problem, and no active input is detected, the IC turns the output off and declares that there was a problem with the FeedBack (FB) of the signal. If this is an initial trial with your ELM327, this is almost certainly a wiring problem. Check your wiring before proceeding.

LV RESET

The ELM327 continually monitors the 5V supply to ensure that it is within acceptable limits. If the voltage should go below the low limit, a 'brownout reset' circuit is activated, the IC stops all activity, performs a full reset, and then prints LV RESET. Note that this type of reset is exactly the same as an AT Z or MCLR reset (but it does not print ELM327 v1.3a).

Beginning with v1.3a of this IC, an LV RESET will also block any automatic searches through the CAN protocols, if bit 4 of PP 2A is a '1' (it is by default). This is done because most LV RESETs will be as a result of CAN wiring problems, if they should start occurring suddenly. Blocking of the CAN protocols is only done until the next reset (AT Z, WS, etc.) or until an AT FE is sent.

NO DATA

The IC waited for the period of time that was set by AT ST, and detected no response from the vehicle. It may be that the vehicle had no data to offer for that particular PID, 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 so that the response was ignored, even though one was sent. If you are certain that there should have been a response, try increasing the ST time (to be sure that you have allowed enough time for the ECU to respond), 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, with an incorrect baud rate setting, or if monitoring and there are messages found that are not being acknowledged, or that contain bit errors. The entire message will be displayed as it was received (if you have filters set, the received mesage may not agree with the filter setting). Try a different protocol, or a different baud rate.

STOPPED

If a protocol initiation or search should be interrupted by a received RS232 character or a low level on the RTS pin, the ELM327 will print the word STOPPED, to show why it is aborting the OBD command, and returning to the prompt state. If you should see this response, then something has interrupted the process that was in progress.

UNABLE TO CONNECT

The ELM327 has 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.

Example Applications

The SAE J1962 standard dictates that all OBD compliant vehicles must provide a standard connector near the driver's seat, the shape and pinout of which is shown in Figure 8 below. The circuitry described here can be used to connect to this J1962 plug without modification to your vehicle.

The male J1962 connector required to mate with a

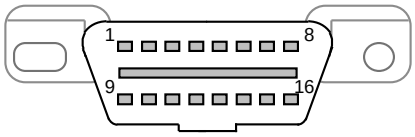


Figure 8. The J1962 Vehicle Connector

vehicle's connector may be difficult to obtain in some locations, and you could be tempted to improvise by making your own connections to the back of your vehicle's connector. If doing so, we recommend that you do nothing that would compromise the integrity of your vehicle's OBD network. The use of any connector which could easily short pins (such as an RJ11 type telephone connector) is definitely not recommended.

The circuit on page 56 (Figure 9) shows how the ELM327 might typically be used. Circuit power is obtained from the vehicle via OBD pins 16 and 5 and, after a protecting diode and some capacitive filtering, is presented to a five volt regulator. (Note that a few vehicles have been reported to not have a pin 5 – on these you will use pin 4 instead of pin 5.) The regulator powers several points in the circuit as well as an LED (for visual confirmation that power is present). We have shown a 78L05 for the regulator as that limits the current available to about 100mA which is a safe value for experimenting. The CAN interface is a low impedance circuit however, and if doing sustained transmissions on CAN, this type of regulator may cause LV RESETs or possibly shut down on over-temperature. Should you experience either of these problems, you may want to consider using a 1 Amp version of the regulator (ie 7805).

The top left corner of Figure 9 shows the CAN interface circuitry. We do not advise making your own interface using discrete components – CAN buses may have a lot of critical information on them, and you can easily do more harm than good if you fail. It is strongly recommended that you use a commercial transceiver chip as shown. The Microchip MCP2551 is used in our circuit, but most major manufacturers produce CAN transceiver ICs – look at the NXP 82C251 (NXP was formerly Philips), the Texas Instruments SN65LBC031, and the Linear Technology LT1796, to name only a few. Be sure to pay attention to the voltage limits – depending on the application, you may have to tolerate 24V, not just 12V.

The next interface shown is for the ISO 9141 and ISO 14230 connections. We provide two output lines, as required by the standards, but depending on your vehicle, you may not need to use the ISO-L output. (Many vehicles do not require this signal for initiation, but some do, so it is shown here.) If your vehicle does not require the L line, simply leave pin 22 unused.

The ELM327 controls both of the ISO outputs through NPN transistors Q6 and Q7 as shown. These transistors have 510Ω pullup resistors connected to their collectors, as the standard requires. We are often asked about substitutes for these resistors – if you need to substitute, you can either go up to 560Ω or possibly make 510Ω from two resistors (1/4W 240Ω + 270Ω resistors work well), but we do not recommend using a lower value as it stresses every device on the bus. Note that 1/2W resistors should be used as a short at 13.8V causes about 0.4W dissipation.

Data is received from the K Line of the OBD bus and connected to pin 12 after being reduced by the R20/R21 voltage divider shown. Because of the Schmitt trigger input on pin 12, these resistors will give typical threshold levels of 9.1V (rising) and 4.7V (falling), providing a large amount of noise immunity while also protecting the IC.

The final OBD interface shown is for the two J1850 standards. The J1850 VPW standard needs a positive supply of up to 8V while the J1850 PWM needs 5V, so we have shown a two level supply that can provide both. This dual voltage supply uses a 317L adjustable regulator as shown, controlled by the pin 3 output. With the resistor values given, the selected voltages will be about 7.5V and 5V, which works well for most vehicles. The two J1850 outputs are driven by the Q1-Q2 combination for the Bus+, and Q3 for the Bus-.

The J1850 VPW input uses a resistor divider as was used for the ISO input. Typical threshold voltages with the resistors shown will be about 4.2V (rising) and 2.2V (falling). The J1850 PWM input is a little different in that it must convert a differential input into a single-ended one for use by the ELM327. In operation, Q4 is actually used as the difference amplifier. The Q4-D3

Example Applications (continued)

series circuit sets a threshold voltage of about 1V (for noise immunity), while R11 limits the current flow, and R12 keeps Q4 off when the input is left open-circuited.

Resistor R36 has been added to the circuit of Figure 9, to help turn transistor Q4 off more rapidly in certain circumstances. The resistor is generally not required, but it may be helpful if you are connected to a very high capacitance J1850 VPW system then force the ELM327 to operate in the J1850 PWM mode, and experience some false BUS ERRORs. We show the resistor as an option and leave the choice whether to install it up to you.

The voltage monitoring circuitry for the AT RV command is shown in this schematic connected to pin 2 of the ELM327. The two resistors simply divide the battery voltage to a safe level for the ELM327, and the capacitor filters out noise. As shipped, the ELM327 expects a resistor divider ratio as shown, and sets nominal calibration constants assuming that. If your application needs a different range of values, simply choose the resistor values to maintain the input within the specified 0-5V limit, and then perform an AT CV to calibrate the ELM327 to your new divider ratio. The maximum voltage that the ELM327 can show is 99.9V.

A very basic RS232 interface is shown connected to pins 17 and 18 of the ELM327. This circuit 'steals' power from the host computer in order to provide a full swing of the RS232 voltages without the need for a negative supply. The RS232 pin connections shown are for a standard 9 pin connector. If you are using a 25 pin one, you will need to compensate for the differences. The polarity of the ELM327's RS232 pins is such that they are compatible with standard interface ICs (MAX232, etc.), so if you should prefer such an interface, you can remove all of the discrete components shown and use one of those.

The four LEDs shown (on pins 25 to 28) have been provided as a visual means of confirming circuit activity. They are not essential, but it is nice to see the visual feedback when experimenting.

Finally, the crystal shown connected between pins 9 and 10 is a standard 4.000MHz microprocessor type crystal. The 27pF crystal loading capacitors shown are typical only, and you may have to select other values depending on what is specified for the crystal you obtain. The crystal frequency is critical to circuit operation and must not be altered.

We often receive requests for parts lists to accompany our Example Applications circuits. Since this circuit is more complex than most, we have named/numbered all of the components and provided a summary parts list (see Figure 10). Note that these are only suggestions for parts. If you prefer another LED colour, or have a different general purpose transistor on hand, etc., by all means make the change. A quick tip for those having trouble finding a 0.3" wide socket for the ELM327: many of the standard 14 pin sockets can be placed end-to-end to form one 0.3" wide 28 pin socket.

What if you only want to use one of the protocols? What if you want to use a USB interface? These are common questions that we receive, and both are addressed in Figure 11.

There are a few single IC products on the market that allow you to connect an RS232 system directly to USB. We have shown the CP2102 by Silicon Laboratories (http://www.silabs.com) in Figure 11, but there are others available as well – Future Technology Devices (http://www.ftdichip.com), for example, produces several. These ICs provide a very simple and relatively inexpensive way to 'bridge' between RS232 and USB, and as you can see, require very few components to support them.

If using the CP2102, we do caution that it is very small and difficult to solder by hand, so be prepared for that. Also, if you provide protection on the data lines with transient voltage suppressors (TVS's), be careful of which ones you choose, as some exhibit a very large capacitance and will affect the transmission of the USB data. Note also that the circuit as presented will operate at a 38400 bits per second rate. If you want to take full advantage of the speed of the USB interface, you will need to change PP 0C.

Considering the OBD protocol portions of the circuits in Figures 9 and 11, the differences should be very apparent. The unused protocols in Figure 11 have simply had their outputs ignored (left open circuit), and their inputs wired to a convenient logic level (as CMOS inputs must never be left floating).

The circuit maintains the status LEDs, and the J1850 Bus+ circuitry, but the majority of the rest has been eliminated. The voltage switching circuitry has been reduced to a single 8V regulator as well, since there will be no need to switch to 5V. Note that pin 3 has been intentionally left open-circuited as it is not required by the voltage regulator.

The first time that this circuit is used, it will likely be set to protocol 0, the default 'automatic search' mode of operation (as shipped from the factory). When you connect it to the J1850 VPW vehicle, it will then

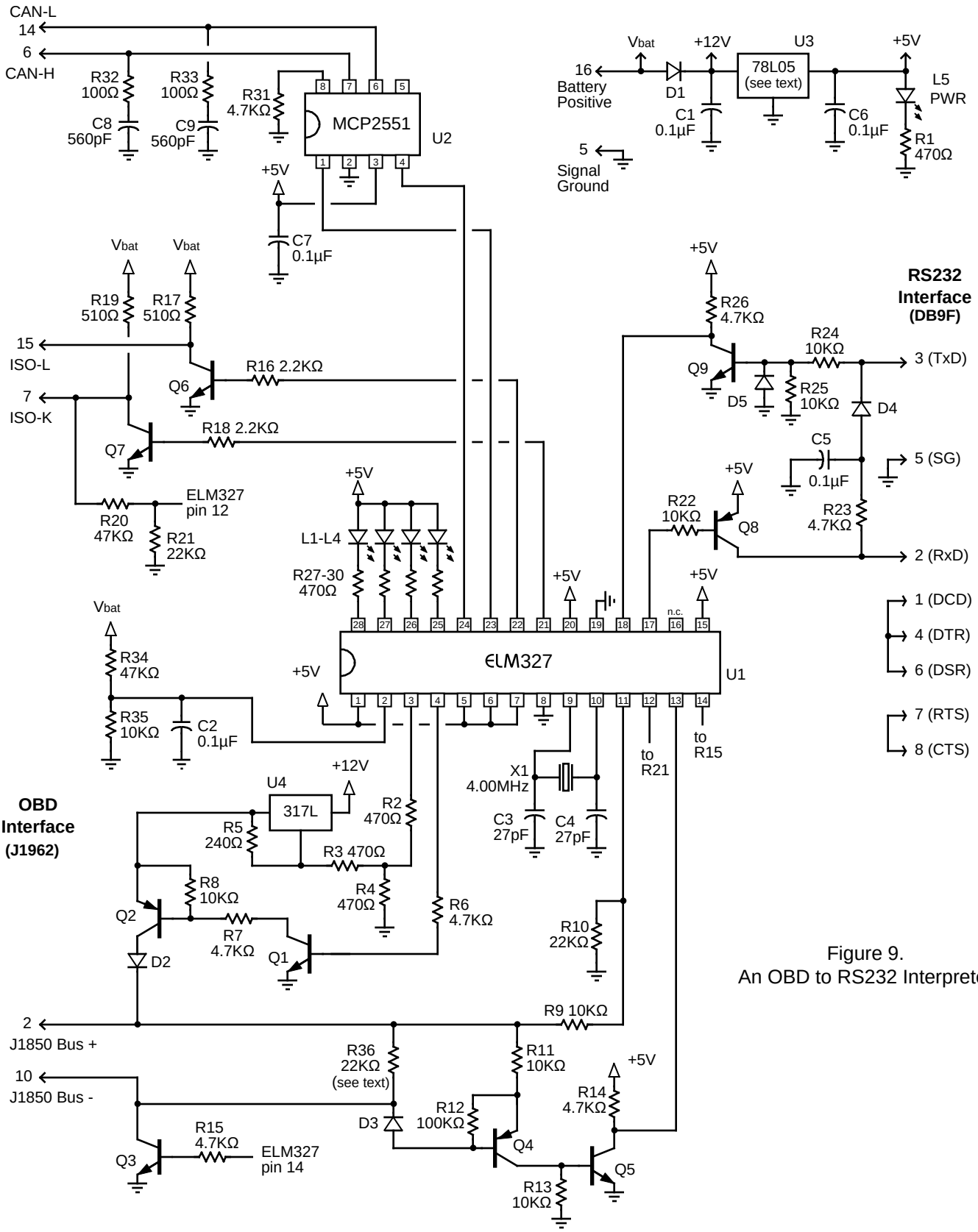


Figure 9.
An OBD to RS232 Interpreter

Example Applications (continued)

automatically detect the protocol, and if the memory is enabled (as shown), J1850 VPW will then become the new default, with no input required by you. This will work well for most applications, but if the circuit is used on a vehicle with the key off, for example, then it will again go searching for a new protocol. In general, you do not want this to happen every time. It may be only a minor inconvenience to have to wait while the ELM327 determines that it is "UNABLE TO CONNECT", but why go through it if you do not have to? If you know that you are using the circuit in a J1850 VPW only application (protocol 2) then you should issue the command AT SP 2 the very first time that the circuit is powered. From that point on, it will remain in protocol 2, whether it fails to make a connection or not.

Depending on the circumstances, you may be able to simplify this circuit even further, by possibly using the USB connection to obtain 5V for the ELM327, rather than the 78L05 regulator shown. We caution that some protocols (CAN for example), may draw more current than your USB connection is able to supply, so review this first.

This has provided two examples of how the ELM327 integrated circuit might be used. Hopefully it has been enough to get you started on your way to many more… we hope that you enjoy the journey!

Semiconductors	Resistors
D1 = 1N4001	R32, R33= 100 Ω
D2, D3, D4, D5 = 1N4148	R5 = 240 Ω
L1, L2, L3, L4 = Yellow LED	R1, R2, R3, R4, R27, R28, R29, R30 = 470 Ω
L5 = Green LED	R17, R19 = 510 Ω 1/2W
Q1, Q3, Q5, Q6, Q7, Q9 = 2N3904 (NPN)	R16, R18 = 2.2 KΩ
Q2, Q4, Q8 = 2N3906 (PNP)	R6, R7, R14, R15, R23, R26, R31 = 4.7 KΩ
U1 = ELM327	R8, R9, R11, R13, R22, R24, R25, R35 = 10 KΩ
U2 = MCP2551	R10, R21, R36 = 22 KΩ
U3 = 78L05 (5V, 100mA regulator)	R20, R34 = 47 KΩ
U4 = 317L (adj. 100mA regulator)	R12 = 100 KΩ
Capacitors	**Misc**
C1, C2, C5, C6, C7 = 0.1uF 16V	X1 = 4.000MHz crystal
C3, C4 = 27pF	RS232 Conn = DB9F
C8, C9 = 560pF	IC Socket = 28pin 0.3" (or 2 x 14pin)

Figure 10. Parts List for Figure 9

Example Applications (continued)

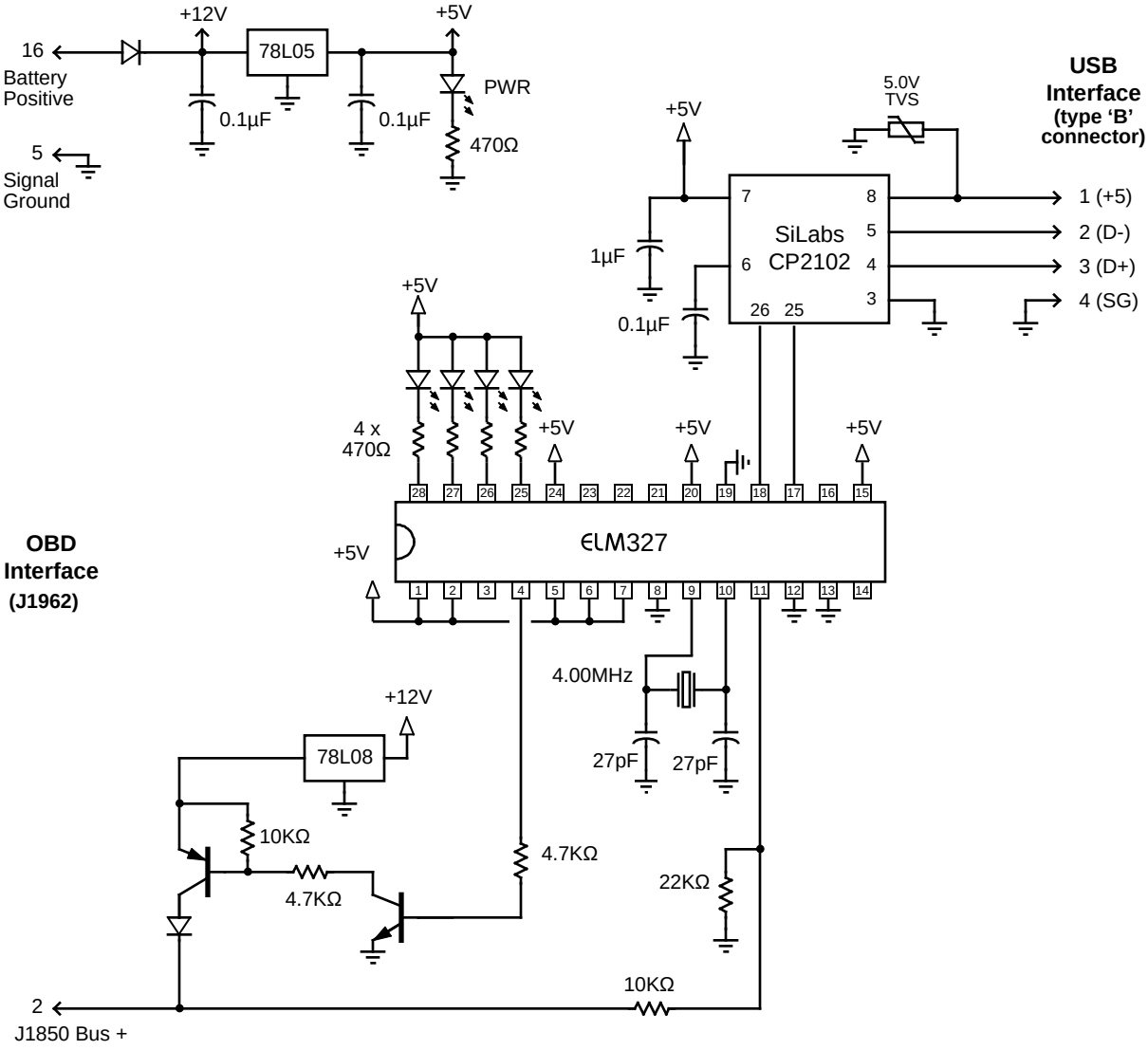


Figure 11. An OBD (J1850 VPW) to USB Interpreter

Section Index

Description and Features..... 1
Pin Descriptions..... 2
Ordering Information..... 3
Absolute Maximum Ratings.....4
Electrical Characteristics.....4
Overview..... 5
Communicating with the ELM327.....5
AT Commands..... 7
AT Command Summary.....7
AT Command Descriptions..... 9
Reading the Battery Voltage..... 22
OBD Commands..... 23
Talking to the Vehicle.....24
Interpreting Trouble Codes.....26
Resetting Trouble Codes.....27
Quick Guide for Reading Trouble Codes..... 27
Bus Initiation.....28
Wakeup Messages.....28
Selecting Protocols.....29
OBD Message Formats.....30
Setting the Headers.....32
Monitoring the Bus..... 35
CAN Messages and Filtering.....36
Multiline Responses..... 37
CAN Message Formats.....39
Altering Flow Control Messages.....40
J1939 Support.....41
Restoring Order.....43
Programmable Parameters..... 44
Programmable Parameter Summary.....45
Using Higher RS232 Baud Rates.....48
Setting Timeouts - AT ST and AT AT Commands..... 50
Microprocessor Interfaces.....51
Error Messages.....52
Example Applications.....54