ELM327

OBD to RS232 Interpreter

Description

Almost all of the automobiles produced today are required, by law, to provide an interface for the connection of diagnostic test equipment. The data transfer on these interfaces follow several standards, but none of them are directly usable by PCs or smart devices. The ELM327 is designed to act as a bridge between these On-Board Diagnostics (OBD) ports and a standard RS232 serial interface.

In addition to being able to automatically detect and interpret nine OBD protocols, the ELM327 also provides support for high speed communications, a low power sleep mode, and the J1939 truck and bus standard. It is also completely customizable, should you wish to alter it to more closely suit your needs.

The following pages discuss all of the ELM327's features in detail, how to use it and configure it, as well as providing some background information on the protocols that are supported. There are also schematic diagrams and tips to help you to interface to microprocessors, construct a basic scan tool, and to use the low power mode.

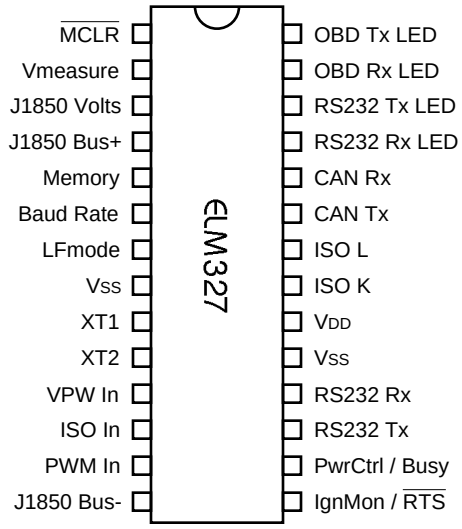
Applications

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

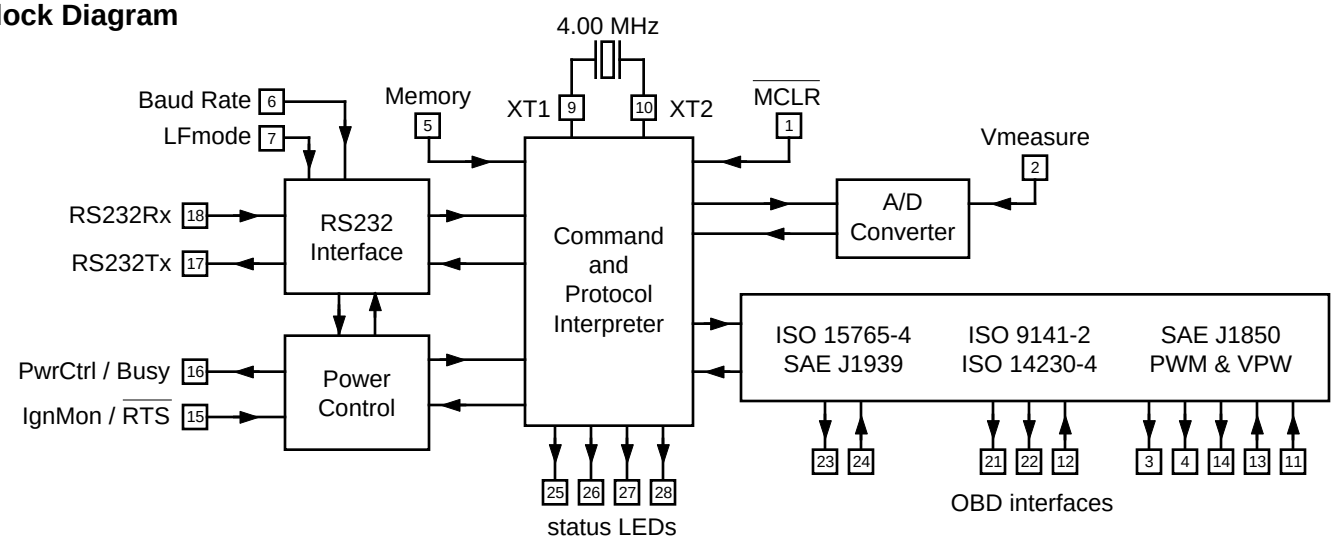
Features

- Power Control with standby mode
- Universal serial (RS232) interface
- Automatically searches for protocols
- Fully configurable with AT commands
- Low power CMOS design

Connection Diagram
PDIP and SOIC
(top view)



Block Diagram



Contents

Contents

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 normally outputs a logic high level when a nominal 8V is required (for J1850 VPW), and a low level for 5V (for J1850 PWM), but this can be changed with PP 12. 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 be initialized to 9600 bps.

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} (pin 8)

Circuit common must be connected to this pin.

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. Also, note that this oscillator is turned off when in the Low Power or ‘standby’ mode of operation.

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

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 wave shaping, 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.

Pin Descriptions (continued)

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

This input pin can serve one of two functions, depending on how the Power Control options (PP 0E) are set.

If both bit 7 and bit 2 of PP 0E are '1's, this pin will act as an Ignition Monitor. This will result in a switch to the Low Power mode of operation, should the IgnMon signal go to a low level, as would happen if the vehicle's ignition were turned off. An internal 'debounce' timer is used to ensure that the ELM327 does not shut down for noise at the input.

When the voltage at pin 15 is again restored to a high level, and a time of 1 or 5 seconds (as set by PP 0E bit 1) passes, the ELM327 will perform a 'Warm Start' and return to normal operation. A low to high transition at pin 15 will in fact restore normal operation, regardless of the setting of PP 0E bit 2, or whether pin 15 was the initial cause for the low power mode. This feature allows a system to control how and when it switches to low power standby operation, but still have automatic wakeup by the ignition voltage, or even by a pushbutton.

If either bit 7 or bit 2 of PP 0E are '0', this pin will function as an active low 'Request To Send' input. This can be used to interrupt the OBD processing in order to send a new command, or as previously mentioned, to highlight the fact that the ignition has been turned off. Normally kept at a high level, this input is brought low for attention, and should remain so until the Busy line (pin 16) indicates that the ELM327 is no longer busy, or until a prompt character is received (if pin 16 is being used for power control).

This input has Schmitt trigger wave shaping. By default, pin 15 acts as the $\overline{\text{RTS}}$ interrupt input.

PwrCtrl / Busy (pin 16)

This output pin can serve one of two functions, depending on how the Power Control options (PP 0E) are set.

If bit 7 of PP 0E is a '1' (the default), this pin will function as a Power Control output. The normal state of the pin will be as set by PP 0E bit 6, and the pin will remain in that state until the ELM327 switches to the Low Power mode of operation, when the output changes to the opposite level. This output is typically used to control enable inputs, but may also be used for relay circuits, etc. with suitable buffering. The discussion on page 85 ('Modifications for Low Power Standby Operation') provides more detail on how to use this output.

If bit 7 of PP 0E is a '0', pin 16 will function as a 'Busy' output, showing when the ELM327 is actively processing a command (the output will be at a high level), or when it is idle, ready to receive commands (the output will be low).

By default, bit 7 of PP 0E is '1', so pin 16 provides the Power Control function.

RS232Tx (pin 17)

This is the RS232 data transmit output. The signal level is compatible with most interface ICs (the output is high when idle), 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 should be high), but can be used with other interfaces as well, since the input has Schmitt trigger wave shaping.

V_{SS} (pin 19)

Circuit common must be connected to this pin.

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 ELM327 processor, 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 transceiver IC (see the

Pin Descriptions (continued)

Example Applications section for more information). If unused, pin 24 must 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, if you can not do so by using the normal interface - see page 70 for details.

Unused Pins

When people only want to implement a portion of what the ELM327 is capable of, they often ask what to do with the unused pins. The rule is that unused outputs may be left open-circuited with nothing connected to them, but unused inputs must be terminated. The ELM327 is a CMOS integrated circuit that can not have any inputs left floating (or you might damage the IC). Connect unused inputs as follows:

Pin	1	2	5	6	7	11	12	13	15	18	24
Level	H	H*	H*	H*	H*	H*	L*	L*	H	H	H

Note that the inputs that are shown with an asterisk (*) may be connected to either a High (V_{DD}) or a Low (V_{SS}) level, but the level shown is preferred.

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}		4.2	5.0	5.5	V	
V_{DD} rate of rise		0.05			V/ms	see note 2
Average current, I_{DD}	normal		12		mA	ELM327 device only - does not include any load currents
	low power		0.15		mA	
Input logic levels	low	V_{SS}		0.8	V	Pins 5, 6, 7, and 24 only
	high	3.0		V_{DD}	V	
Schmitt trigger input thresholds	rising		2.9	4.0	V	Pins 1, 11, 12, 13, 15 and 18 only
	falling	1.0	1.5		V	
Output low voltage			0.3		V	current (sink) = 10 mA
Output high voltage			4.4		V	current (source) = 10 mA
Brown-out reset voltage		2.65	2.79	2.93	V	
A/D conversion time			9		msec	AT RV to beginning of response
Pin 18 wake pulse duration		128			µsec	to wake from Low Power mode
IgnMon debounce time		50	65		msec	
AT LP to PwrCtrl output time			1.0		sec	
LP ALERT to PwrCtrl output time			2.0		sec	
Reset time	AT Z		800		msec	Measured from the end of the command to the start of the ID message (ELM327 v2.2)
	AT WS		2		msec	

Notes:

1. This integrated circuit is based on Microchip Technology Inc.'s PIC18F2480 device. For more detailed device specifications, and possibly clarification of those given, please refer to the Microchip documentation (available at 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.

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 integrated circuit 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 smart device with a terminal program (such as HyperTerminal or ZTerm), and a little knowledge of 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 adjust 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 v2.2

>
```

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 that you send 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.), 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 the request to the vehicle - simply send a carriage return character, and the ELM327 will repeat the command for you. The memory buffer only remembers one command though, and 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 ignore incoming bytes 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 31) may be helpful if you wish to interpret the values. Also, you 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 two pages following provide a summary of all of the commands that the current version of the ELM327 recognizes. A more complete description of each command begins on page 12. Note that the settings which are shown with an asterisk (*) are the default values.

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
LP	go to Low Power mode
M0, M1	Memory off, or on
RD	Read the stored Data byte
SD hh	Save Data byte hh
WS	Warm Start (quick software reset)
Z	reset all
@1	display the device description
@2	display the device identifier
@3 ccccccccccc	store the @2 identifier

Programmable Parameter Commands

PP xx OFF	disable Prog Parameter xx
PP FF OFF	all Prog Parameters disabled
PP xx ON	enable Prog Parameter xx
PP FF ON	all Prog Parameters enabled
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
CV 0000	restore CV value to factory setting
RV	Read the input Voltage

Other

IGN	read the IgnMon input level

AT Command Summary (continued)

OBD Commands

AL	Allow Long (>7 byte) messages
AMC	display Activity Monitor Count
AMT hh	set the Activity Mon Timeout to hh
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
SH wwxxyyzz	Set Header to wwxxyyzz
SP h	Set Protocol to h and save it
SP Ah	Set Protocol to Auto, h and save it
SP 00	Erase stored protocol
SR hh	Set the Receive address to hh
SS	use Standard Search order (J1978)
ST hh	Set Timeout to hh x 4 msec
TA hh	set Tester Address to hh
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, if not monitoring
IFR4, 5, 6	IFRs off, auto, or on, at all times
IFRH, S	IFR value from Header* or Source

ISO Specific Commands (protocols 3 to 5)

FI	perform a Fast Initiation
IB10	set the ISO Baud rate to 10400*
IB12	set the ISO Baud rate to 12500
IB15	set the ISO Baud rate to 15625
IB48	set the ISO Baud rate to 4800
IB96	set the ISO Baud rate to 9600
IIA hh	set ISO (slow) Init Address to hh
KW	display the Key Words
KW0, KW1	Key Word checking off, or on*
SI	perform a Slow (5 baud) Initiation
SW hh	Set Wakeup interval to hh x 20 msec
SW 00	Stop sending Wakeup messages
WM [1 - 6 bytes]	set the Wakeup Message

CAN Specific Commands (protocols 6 to C)

CAF0, CAF1	Automatic Formatting off, or on*
CEA	turn off CAN Extended Addressing
CEA hh	use CAN Extended Address hh
CER hh	set CAN Extended Rx address to hh
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	reset the Receive Address filters
CRA hhh	set CAN Receive Address to hhh
CRA hhhhhhhh	set the Rx Address to hhhhhhhh

AT Command Summary (continued)

CAN Specific Commands (continued)

CS	show CAN Status counts & freq
CSM0, CSM1	Silent Monitoring off, or on*
CTM1	set Timer Multiplier to 1*
CTM5	set Timer Multiplier to 5
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	Set the Header to hhhhhhhh
FC SD [1 - 5 bytes]	FC, Set Data to [...]
PB xx yy	Protocol B options and baud rate
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*
JHF0, JHF1	Header Formatting off, or on*
JS	use J1939 SAE data format
JTM1	set Timer Multiplier to 1*
JTM5	set Timer Multiplier to 5
MP hhhh	Monitor for PGN 0hhhh
MP hhhh n	" " and get n messages
MP hhhhhh	Monitor for PGN hhhhhh
MP hhhhhh n	" " and get n messages

AT Command Descriptions

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

<CR> [repeat the last command]

Sending a single carriage return character causes the ELM327 to repeat the last command that it performed. This is typically used when you wish to obtain updates to a value at the fastest possible rate - for example, you may send 01 0C to obtain the engine rpm, then send only a carriage return character each time you wish to receive an update.

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).

AMC [display Activity Monitor Count]

The Activity Monitor uses a counter to determine just how active the ELM327's OBD inputs are. Every time that activity is detected, this counter is reset, while if there is no activity, the count goes up (every 0.655 seconds). This count then represents the time since activity was last detected, and may be useful when writing your own logic based on OBD activity. The counter will not increment past FF (internal logic stops it there), and stays at 00 while monitoring.

AMT hh [set the Act Mon Timeout to hh]

When the Activity Monitor Count (ie time) exceeds a certain threshold, the ELM327 decides that there is no OBD activity. It might then give an ACT ALERT message or switch to Low Power operation, depending on how the bits of PP 0F are set. The threshold setting is determined by either PP 0F bit 4, or by the AT AMT value, should you provide it. The actual time to alarm will be (hh+1) x 0.65536 seconds. Note that a value of 00 is accepted for AMT, but is used to block all Activity Monitor outputs.

AR [Automatically set the Receive address]

Responses from the vehicle will be acknowledged and displayed by the ELM327, if the internally stored receive address matches the address that the

AT Command Descriptions (continued)

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 protocol.

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 automatically sets 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 (so the timeout is always as set by AT ST), while AT2 is a more aggressive 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 – it uses 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' (ie 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. For example, here's one 'dump':

```
>AT BD
05 C1 33 F1 3E 23 C4 00 00 10 F8 00 00
```

The 05 is the length byte - it tells us that only the first 5 bytes (ie C1 33 F1 3E and 23) are valid. The remaining bytes are likely left over from a previous operation.

The length byte always 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 it represents the actual 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, while under computer control. It is not intended for casual experimenting - if you wish to change the baud rate from a terminal program, you should use PP 0C.

Since some interface circuits are not able to operate at high data rates, the BRD command uses a sequence of sends and receives to test the interface, with any failure resulting in a fallback to the previous baud rate. This allows several baud rates to be tested and a reliable one chosen for the communications. The entire process is described in detail in the 'Using Higher RS232 Baud Rates' section, on pages 51 and 52.

If successful, the actual baud rate (in kbps) will be 4000 divided by the divisor (hh). The value 00 is not accepted by the BRD command.

AT Command Descriptions (continued)

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 75 msec. 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 formatting (PCI) bytes will be automatically generated for you when sending, and will be removed 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 are all shown, with a '<DATA ERROR' message beside them).

Multi-frame responses may be returned by the vehicle with ISO 15765 and SAE J1939. To make these more readable, the Auto Formatting mode will extract the total data length and print it on one line, then show each line of data with the segment number followed by a colon (':'), and then the data bytes.

You may also see the characters 'FC:' on a line (if you are experimenting). This identifies a Flow Control message that has been 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 if auto formatting is on or not).

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 data bytes as received. No bytes will be hidden from you, and none will be inserted for you. Similarly, when sending data with formatting off, you must provide all of the required data bytes exactly as they are to be sent – the ELM327 will not add a PCI byte for you (but it will add some trailing 'padding' bytes to ensure that the required eight data bytes are sent). This allows the ELM327 to be used with protocols that have special formatting requirements.

Note that turning the display of headers on (with AT H1) will override some of the CAF1 formatting of the received data, 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.

CEA [turn off the CAN Extended Address]

The CEA command is used to turn off the special features that are set with the CEA hh command.

CEA hh [set the CAN Extended Address to hh]

Some (non-OBD) CAN protocols extend the addressing fields by using the first of the eight data bytes as a target (receiver) address. This command allows the ELM327 to interact with those protocols.

Sending the CEA hh command causes the ELM327 to insert the hh value as the first data byte of all CAN messages that you send. It also adds one more filtering step to received messages, only passing ones that have the Tester Address in the first byte position (in addition to requiring that ID bits match the patterns set by AT CF and CM, or CRA). The AT CEA hh command can be sent at any time, and changes are effective immediately, allowing for changes of the address 'on-the-fly'. There is a more lengthy discussion of extended addressing in the 'Using CAN Extended Addresses' section on page 62.

The CEA mode of operation is off by default, and is enabled by sending the CEA command with a target address. Once it is on, it can be turned off by sending AT CEA (with no address), or by restoring the chip defaults with AT D, AT Z, etc. Note that the CEA setting has no effect when J1939 formatting is on.

CER hh [set the CAN Extended Rx address to hh]

By default, the ELM327 receives responses to CAN extended addressing requests that have the 'tester address' in the first data byte position. The CER command allows you to choose a different receive address.

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 may be given any value. 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 CAN protocol expects a 'Flow Control' message to always be sent in response to a 'First Frame' message, and 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.

As of firmware version 2.0, these commands also enable or disable the sending of J1939 TP.CM_CTS messages in response to TP.CM_RTS requests.

During monitoring (AT MA, MR, or MT), there are never any Flow Controls sent no matter what the CFC option is set to. The default setting is CFC1 - Flow Controls on.

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 signalled by setting a mask bit to '1', while a 'don't care' is signalled 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 as 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 assign the five most significant bits of the 29 bit CAN ID that is used for sending messages (the other 24 bits are set with the AT SH command). Many systems use these bits to assign a priority value to messages, and to determine the protocol. Any bits provided in excess of the five required are 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, which can be restored at any time with the AT D command.

CRA [reset the CAN Rx Addr]

The AT CRA command is used to restore the CAN receive filters to their default values. Note that it does not have any arguments (ie no data).

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

AT Command Descriptions (continued)

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.

If you wish to allow the reception of a range of values, you can use the letter X to signify a 'don't care' condition. That is, AT CRA 7EX would allow all IDs that start with 7E to pass (7E0, 7E1, etc.). For a more specific range of IDs, you may need to assign a mask and filter.

To reverse the changes made by the CRA command, simply send AT CRA or AT AR.

CRA hhhhhhhh [set the CAN Rx Addr to hhhhhhhh]

This command is identical to the previous one, except that it is used with 29 bit CAN IDs. Sending either AT CRA or AT AR will also reverse any 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.

Beginning with firmware v2.2, the CS response will also show the frequency of the signal at the CAN input, at that time. A typical response might look like:

```
>AT CS
T:00 R:00 F:250
```

The same module that determines the frequency during protocol searches is used, and is fairly basic in operation. It provides frequency in only certain ranges, so possible responses with the current version are limited to 500, 250, <250, and 0 (no signal).

CSM0 and **CSM1** [CAN Silent Monitoring off or on]

The ELM327 was designed to be completely silent while monitoring a CAN bus. Because of this, it is able to report exactly what it sees, without colouring the information in any way. Occasionally (when bench testing, or when connecting to a dedicated CAN port), it may be preferred that the ELM327 does not operate silently (ie generates ACK bits, etc.), and this is what the CSM command is for. CSM1 turns it on, CSM0 turns it off, and the default value is determined by PP 21. Be careful when experimenting with this. If you should choose the wrong baud rate then monitor the CAN bus with the silent monitoring off, you will disturb the flow of data. Always keep the silent monitoring on until you are certain that you have chosen the correct baud rate.

CTM1 [set the Timer Multiplier to 1]

This command causes all timeouts set by AT ST to be multiplied by a factor of 1. Note that this currently only affects the CAN protocols (6 to C). CTM1 is the default setting.

CTM5 [set the Timer Multiplier to 5]

This command causes all timeouts set by AT ST to be multiplied by a factor of 5. Note that this currently only affects the CAN protocols (6 to C).

This command was originally added (as JTM5) to assist with the retrieving of some J1939 messages. We have since had several requests to allow it to affect all CAN modes, and so have modified the JTM5 code and added the new CTM1/CTM5 commands. If using CTM5, we caution that the Adaptive Timing code does not monitor changes in the setting, so we advise turning it off (with AT AT0).

By default, this multiplier is off.

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 30 for some more information on how to read voltages and perform the calibration.

CV 0000 [restore the factory Calibration Value]

If you are experimenting with the CV dddd

AT Command Descriptions (continued)

command but do not have an accurate voltmeter as a reference, you may soon get into trouble. If this happens, you can always send AT CV 0000 to restore the ELM327 to the factory calibration value.

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 of DLC off or on]

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, by way of 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 automatically detects a vehicle's OBD protocol, but does not normally report what it is. The DP command is a convenient means of asking what protocol the IC is currently set to (even if it has not yet 'connected' to the vehicle).

If a protocol is chosen and the automatic option is also selected, AT DP will show the word 'AUTO' before the protocol description. Note that the description shows the actual protocol names, 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.

E0 and **E1** [Echo off or on]

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

The data bytes that are sent in a CAN Flow Control message may be defined with this command. One to five data bytes may be specified, with the remainder of the data bytes in the message being automatically set to the default CAN filler byte, if required by the protocol. Data provided with this command is only used when Flow Control modes 1 or 2 have been enabled.

FC SH hhh [Flow Control Set Header to…]

The header (or more properly 'CAN ID') bytes used for CAN Flow Control 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

AT Command Descriptions (continued)

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 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. Note that FC modes 1 and 2 can only be enabled if you have defined the needed data and possibly ID bytes. If you have not, you will get an error. More complete details and examples can be found in the Altering Flow Control Messages section (page 61).

FE [Forget Events]

There are certain events which may change how the ELM327 responds from that time onwards. One of these is the occurrence of a fatal CAN error (ERR94), which 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.

Another example is an 'LV RESET' event which will prevent searches through CAN protocols if PP 2A bit 4 is '1'. It may also be forgotten with the AT FE command.

FI [perform a Fast Initiation]

One version of the Keyword protocol uses what is known as a 'fast initiation' sequence to begin communications. Usually, this sequence is performed when the first message needs to be sent, and then the message is sent immediately after. Some ECUs may need more time between the two however, and having a separate initiation command allows you to control this time. Simply send AT FI, wait a little, then send the message. You may need to experiment to get the right amount of delay.

Another use for this command might be if you would like to perform a fast initiation with an ISO 9141 type protocol (ie 3 - CARB format). Simply follow these steps to generate a fast init, then switch to protocol 3:

```
AT SP 5
AT FI
AT SP 3
AT BI
```

You should then be able to communicate with the ECU. Note that a protocol close (ie AT PC) is not required in the above code, as the ELM327 automatically performs one when you switch protocols.

Protocol 5 must be selected to use the AT FI command, or an error will result.

H0 and **H1** [Headers off or on]

These commands control whether or not the additional (header) bytes of information are shown in the responses from the vehicle. These are not normally shown by the ELM327, but may be of interest (especially if you receive multiple responses and wish to determine what modules they were from).

Turning the headers on (with AT H1) 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 or AT D1. 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 v2.2'). Software can use this to determine exactly which integrated circuit it is talking to, without having to reset the IC.

IB10 [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.

IB12 [set the ISO Baud rate to 12500]

This command is used to change the baud rate used for the ISO 9141-2 and ISO 14230-4 protocols (numbers 3, 4, and 5) to 12500 baud.

IB15 [set the ISO Baud rate to 15625]

This command is used to change the baud rate used for the ISO 9141-2 and ISO 14230-4 protocols (numbers 3, 4, and 5) to 15625 baud.

AT Command Descriptions (continued)

IB48 [set the ISO Baud rate to 4800]

This command is used to change the baud rate used for the ISO 9141-2 and ISO 14230-4 protocols (numbers 3, 4, and 5) to 4800 baud.

IB96 [set the ISO Baud rate to 9600]

This command 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.

IFR0, IFR1, and **IFR2** [IFR control - not monitoring]

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, unless you are monitoring (by default, the ELM327 is always silent while monitoring). You can override this behaviour with the IFR command.

The IFR0 command will disable the sending of all IFRs, no matter what the header bytes require. IFR2 is the opposite - it will cause an IFR byte to always be sent, no matter what the header bytes say. The IFR1 command determines whether to send an IFR from the value of the 'K' bit in the first header byte (for both PWM and VPW). The default setting is IFR1.

IFR4, IFR5, and **IFR6** [IFR control - at all times]

While the original ELM327 ICs would never send IFRs while monitoring, there are certain situations (typically when simulating an ECU and using one ELM327 to continuously receive data while another only sends). In this case, it is desirable to be able to send IFRs while monitoring.

The IFR4, IFR5, and IFR6 commands mimic the IFR0, IFR1 and IFR2 commands respectively, except that they apply at all times (whether monitoring or not). IFR4 will never generate an IFR, IFR5 will do it based on the 'K' bit value, and IFR6 always generate one.

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 (ie tester) address byte that was in the header of the request. There may be occasions when it is desirable 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 or AT TA), even if another value was sent in the Header bytes. This is not what is normally required, and caution should be used when doing this. AT IFR H restores the sending of the IFR bytes to those provided in the Header, and is the default setting.

IGN [read the IgnMon input level]

This command reads the signal level at pin 15. It assumes that the logic level is related to the ignition voltage, so if the input is at a high level, the response will be 'ON', and a low level will report 'OFF'.

This feature is most useful if you wish to perform the power control functions using your own software. If you disable the Low Power automatic response to a low input on this pin (by setting bit 2 of PP 0E to 0), then pin 15 will function as the RTS input. A low level on the input will not turn the power off, but it will interrupt any OBD activity that is in progress. All you need to do is detect the 'STOPPED' message that is sent when the ELM327 is interrupted, and then check the level at pin 15 using AT IGN. If it is found to be OFF, you can perform an orderly shutdown yourself.

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

AT Command Descriptions (continued)

'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 you to provide 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.

JHF0 and **JHF1** [J1939 Header Formatting off or on]

When printing responses, the ELM327 normally formats the J1939 ID (ie Header) bits in such a way as to isolate the priority bits and group all the PGN information, while keeping the source address byte separate. If you prefer to see the ID information as four separate bytes (which a lot of the J1939 software seems to do), then simply turn off the formatting with JHF0. The CAF0 command has the same effect (and overrides the JHF setting), but also affects other formatting. The default setting is JHF1.

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 (ie in the order specified by the SAE documents), then select JS formatting.

As an example, when sending a request for engine temperature (PGN 00FEEE) with the data format set to JS, you must present 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.

JTM1 [set the J1939 Timer Multiplier to 1]

This command is used to set the AT ST time multiplier to x1 for the SAE J1939 protocol. As of firmware v2.1, this command now simply calls the CTM1 command.

JTM5 [set the J1939 Timer Multiplier to 5]

This used to set the AT ST time multiplier to x5, for the SAE J1939 protocol. As of firmware v2.1, this command now simply calls the CTM5 command.

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.

KW0 and **KW1** [Key Word checks off or on]

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 the bytes are not found, the initiation is said to have failed (you might see 'UNABLE TO CONNECT' or perhaps 'BUS INIT: ...ERROR'). This might occur if you are trying to connect to a non-OBD compliant ECU, or perhaps to an older one.

If you wish to experiment with non-standard systems, you may have to tell the ELM327 to perform the initiation sequence, but ignore the contents of the bytes that are sent and received. To do this, send:

```
>AT KW0
```

After turning keyword checking off, the ELM327 will still require the two key word bytes in the response, but will not look at the actual values of the bytes. It will also send an acknowledgement to the ECU, and will wait for the final response from it (but will not stop and report an error if none is received). This may allow you to make a connection in an otherwise 'impossible' situation. Normal behaviour can be returned with AT KW1, which is the default setting.

L0 and **L1** [Linefeeds off or on]

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

AT Command Descriptions (continued)

setting is determined by the voltage at pin 7 during power on (or reset). If the level is high, then linefeeds are on by default; otherwise they will be off.

LP [go to the Low Power mode]

This command causes the ELM327 to shut off all but 'essential services' in order to reduce the power consumption to a minimum. The ELM327 will respond with an 'OK' (but no carriage return) and then, one second later, will change the state of the PwrCtrl output (pin 16) and will enter the low power (standby) mode. The IC can be brought back to normal operation through a character received at the RS232 input or a rising edge at the IgnMon (pin 15) input, in addition to the usual methods of resetting the IC (power off then on, a low on pin 1, or a brownout). See the Power Control section (page 65) for more information.

M0 and **M1** [Memory off or on]

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.

The default setting for the memory function is determined by the voltage level at pin 5 during 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 (unless you turn silent mode off with CSM0), or Wakeup ('keep-alive') messages for the ISO 9141 and ISO 14230 protocols. Monitoring will continue until you stop it with activity on the RS232 input, or the RTS pin.

To stop 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 was in the middle of printing a line, it will first complete that line and then print 'STOPPED', before returning to the command state and sending a prompt character. If it were simply waiting for input, it would return much quicker. Note that the character which stops the monitoring will always be discarded, and will not affect subsequent commands.

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 those 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 may not be what is desired - you may want to reset the masks and filters (with AT AR or AT CRA) first.

All of the monitoring commands (MA, MR and MT) operate by closing the current protocol (an AT PC is executed internally), before configuring the IC for monitoring the data. 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

AT Command Descriptions (continued)

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 desirable 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.

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 preceded 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 PGN information.

MP hhhh n [Monitor for PGN, get n messages]

This is very similar to the above command, but adds the ability to set the number of messages that should be fetched before the ELM327 automatically stops monitoring and prints a prompt character. The value 'n' may be any single hex digit.

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 other bits are ignored.

MP hhhhhh n [Monitor for PGN, get n messages]

This is very similar to the previous command, but it adds the ability to set the number of messages that should be fetched before the ELM327 automatically stops monitoring and prints a prompt character. The value 'n' may be any single hex digit.

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 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'

AT Command Descriptions (continued)

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.

The ELM327 does not require a change to AL to allow longer message lengths for the KWP protocols to be received. You can simply leave the IC set to the default setting of NL, and all of the received bytes will be shown.

PB xx yy [set Protocol B parameters]

This command allows you to change the protocol B (USER1) options and baud rate without having to change the associated Programmable Parameters. This allows for easier protocol changes while testing.

To use this feature, simply set xx to the value for PP 2C, and yy to the value for PP 2D, and issue the command. The next time that the protocol is initialized it will use these values. For example, to set protocol B for use with 500kbps J1939, simply use the command:

```
>AT PB 42 01
```

As another example, assume that you wish to monitor a system that uses 11 bit CAN at 33.3 kbps. If you do not want any special formatting, this means a value of 11000000 or C0 hex for PP 2C, and 15 decimal or 0F hex for PP 2D. Simply send these values to the ELM327 with the command:

```
>AT PB C0 0F
```

then start monitoring with:

```
>AT MA
```

if you see CAN ERRORs, and realize that you wanted an 83.3 kbps baud rate, then close the protocol, and send the new values:

```
>AT PC
OK

>AT PB C0 06
OK

>AT MA
```

Values passed in this way do not affect any that are stored as the 2C and 2D Programmable Parameters, and are lost if the ELM327 is reset. If you want to make your settings persist over power cycles, then you may wish to store them in the Programmable Parameter for CAN protocols USER1 or USER2.

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 70) 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, as there may be a problem with your connection.

PP hh ON [turn Programmable Parameter hh ON]

This command enables Programmable Parameter number hh. Once enabled, any value assigned using

AT Command Descriptions (continued)

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 70) 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 hh SV yy [Prog. Param. hh: 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, with PP hh ON.

PPS [Programmable Parameter Summary]

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

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 used for normal OBD communications, however, as the vehicle may have difficulty if it is expecting an acknowledgement 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 (ie. 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.

The RA command is exactly the same as the SR command, and can be used interchangeably. Note that CAN Extended Addressing does not use this value - it uses the one set by the AT TA command.

RD [Read the Data in the user memory]

The byte value stored with the SD command is retrieved with this command. There is only one memory location, so no address is required.

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 (one that has been sending and receiving messages), as it can not initiate a protocol search. 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 else turn the headers on.

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

AT Command Descriptions (continued)

RV [Read the input Voltage]

This initiates 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 1:5.7 ratio voltage divider (for example, 47KΩ and 10KΩ resistors in series, 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 (the voltage at pin 2 must not exceed Vdd), with an uncalibrated accuracy of typically about 2%. See the 'Reading the Battery Voltage' section for calibration information.

S0 and **S1** [printing of Spaces off or on]

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.

SD hh [Save Data byte hh]

The ELM327 is able to save one byte of information for you in a special nonvolatile memory location, which is able to retain its contents even if the power is turned off. Simply provide the byte to be stored, then retrieve it later with the read data (AT RD) command. This location is ideal for storing user preferences, unit ids, occurrence counts, or other information.

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 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 stored internally 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). If experimenting, it is not necessary but may be better to set the headers after a protocol is active. That way, wakeup messages, etc. that get set on protocol activation will use the default values.

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

AT Command Descriptions (continued)

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.

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

This four byte version of the AT SH command allows setting a complete 29 bit CAN ID in one instruction. Alternatively, AT SP (for the five most significant bits) and AT SH (for the other three bytes) may be used.

SI [perform a Slow Initiation]

Protocols 3 and 4 use what is sometimes called a 5 baud, or slow initiation sequence in order to begin communications. Usually, the sequence is performed when the first message needs to be sent, and then the message is sent immediately after. Some ECUs may need more time between the two however, and having a separate initiation command allows you to control this time. Simply send AT SI, wait a little, then send the message. You may need to experiment a little to get the right amount of delay. Protocol 3 or 4 must be selected to use the AT SI command, or an error will result.

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 ELM327 supports 12 different protocols (two can be user-defined). They 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 - if searching, you may see more than one type of response.

ELM327 users often use the AT SP 0 command to reset the search protocol before starting (or restarting) a connection. This works well, but since it is used so often, and since writes to EEPROM result in an unnecessary delay (of about 30 msec), the AT SP0 command sets the protocol to 0, but does not perform a write to EEPROM. Similarly, the SP A0 and SP 0A commands do not perform writes to EEPROM, either. Saving this value to EEPROM would not provide any advantage (and would be very short-lived, as the ELM327 will soon be finding the vehicle's protocol and over-writing the '0' value in EEPROM). If you really want to store the value '0' in the internal EEPROM, you must use the 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 a response 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 use the one protocol, but is also one that can cause a lot of problems if you do not understand it.

SP 00 [erase the Stored Protocol]

To speed up protocol initiation and detection, the SP 0 command sets the protocol to automatic, but does not perform a (very time-consuming) write to EEPROM. Some users felt it was necessary to be able

AT Command Descriptions (continued)

to actually write to the ELM327's EEPROM, however, so we provided this command. It should not normally be used when connecting to a vehicle.

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 each new protocol detected will become your new default.

SP Ah will save the protocol information even if the memory option is off (but SP A0 and SP 0A do not- if you must write 0 to the EEPROM, use command AT 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 accept a message that was actually meant for another module, possibly sending an IFR when you should not. To turn off the SR filtering, simply send AT AR.

This command has limited use with CAN, as it only monitors one byte of the ID bits, and that is not likely selective enough for most CAN applications (the CRA command may be a better choice). Also, the command has no effect on the addresses used by the J1939 protocols, as the J1939 routines set their own receive addresses based on the ID bit (header) values.

This SR command is exactly the same as the RA command, and can be used interchangeably with it. Note that CAN Extended Addressing does not use this value - it uses the one set by the AT TA command.

SS [use the Standard Sequence for searches]

SAE standard J1978 specifies a protocol search order that scan tools should use. It follows the number order that we have assigned to the ELM327 protocols. In order to provide a faster search, the ELM327 does not normally follow this order, but it will if you command it to with AT SS.

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 may also be used after a response has been received, while waiting to see if any more is coming (but this depends on the AT AT setting). The AT ST command allows this timer to be adjusted, in increments of 4 msec (or 20 msec if in a CAN protocol, with CTM5 selected).

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 longer. In most circumstances, it is best to simply leave the AT ST time at the default setting, and let the adaptive timing algorithm determine what to use for the timeout.

The ST timer is set to 32 by default (giving a time of approximately 200 msec), but this default setting 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. Also, during protocol searches, an internally set minimum time is used - you may select longer times with AT ST, but not shorter ones.

SW hh [Set Wakeup to hh]

Once a data connection has been established, some protocols require that there be data flow every few seconds, just so that the ECU knows to maintain the communications path open. If the messages do not appear, the ECU will assume that you are finished, and will close the channel. The connection will need to be initialized again to reestablish communications.

The ELM327 will automatically generate periodic messages, as required, in order to maintain a connection. Any replies to these messages are ignored by the ELM327, and are not visible to the user. (Currently, only protocols 3, 4, and 5 support these messages - nothing is available for CAN. If you require CAN periodic messages, you must use the ELM329.)

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

AT Command Descriptions (continued)

possible time delay of just over 5 seconds results when a value of FF (decimal 255) is used. The default setting (92) provides a nominal delay of 3 seconds between messages.

Note that the value 00 (zero) is special, as it will stop the periodic (wakeup) messages. This provides a control for experimenters to stop the messages while keeping the rest of the protocol functioning normally, and is not intended to be used regularly. Issuing AT SW 00 will not change a prior setting for the time between wakeup messages, if the protocol is re-initialized. Once periodic messages have been turned off with AT SW, they can only be reestablished by closing and reinitializing the protocol.

TA hh [set the Tester Address to hh]

This command is used to change the current tester (ie. scan tool) address that is used in the headers, periodic messages, filters, etc. The ELM327 normally uses the value that is stored in PP 06 for this, but the TA command allows you to temporarily override that value.

Sending AT TA will affect all protocols, including J1939. This provides a convenient means to change the J1939 address from the default value of F9, without affecting other settings.

Although this command may appear to work 'on the fly', it is not recommended that you try to change this address after a protocol is active, as the results may be unpredictable.

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 EEPROM 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 the other protocols, attempting to connect to one of them.

V0 and **V1** [Variable data lengths off or on]

Many CAN protocols (ie ISO 15765-4) expect to send eight data bytes at all times. The V0 and V1 commands may be used to override this behaviour (for any CAN protocol) should you wish.

Choosing V1 will cause the current CAN protocol to send variable data length messages, just as bit 6 of PP 2C and PP 2E do for protocols B and C. It does not matter what the protocol should be doing - V1 will override that. This allows experimenting with variable data length messages on demand.

If you select V0 (the default setting), the forced sending of variable length CAN messages is turned off. The format of the sent messages reverts to the protocol's settings.

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 message that you wish to have sent (typically three header bytes and one to three data bytes), and the ELM327 will add the checksum and send them as required, at the rate determined by the AT SW setting.

Default settings will send the bytes 68 6A F1 01 00 for ISO 9141, and C1 33 F1 3E for KWP.

WS [Warm Start]

This command causes the ELM327 to perform a complete reset. It is very similar to the AT Z command, but does not include the power on LED test. Users may find this a convenient way 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 into the idle state, waiting for characters on the RS232 bus. Note that 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.

AT Command Descriptions (continued)

@1 [display the device description]

This command displays the device description string. The default text is 'OBDII to RS232 Interpreter'.

@2 [display the device identifier]

A device identifier string that was recorded with the @3 command is displayed with the @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.

See the 'Programming Serial Numbers' section for more information.

@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 you may only use the @3 command one time). The characters sent must be printable (ascii character values 0x21 to 0x5F inclusive).

If you are developing software to write device identifiers, you may be interested in the ELM328 IC, as it allows multiple writes using the @3 command (but it can not send OBD messages).

Reading the Battery Voltage

Before learning 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 81). 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 (' ') within the message either, as they are ignored by the ELM327.

A typical response 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.

Let us assume that you have connected your accurate multimeter, and you find that it reads 12.47V. The ELM327 is a little high at 12.6V, and you would like it to read the same as your meter. Simply calibrate the ELM327 to the measured 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).

The ELM327 may 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
```

If you should get into trouble with this command (for example, if you set calibration values to something arbitrary and do not have a voltmeter on hand to provide accurate values), you can restore the settings to the original (factory) values with the CV 0000 command. Simply send:

```
>AT CV 0000
OK
```

The other AT Commands are used in the same manner. Simply type the letters A and T, then follow with the command you want to send and any arguments that are required. Then press return (or enter, depending on your keyboard). Remember - 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 for you. 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 (usually seven bytes or 14 hexadecimal digits). Attempts to send more bytes will result in an 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. Dealing with the hex digits can not be avoided - eventually all users need to manipulate the results in some way (combining bytes and dividing by 4 to obtain rpm, dividing by 2 to obtain degrees of advance, converting temperatures, etc.).

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 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 standards require that each OBD command or 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) specifies 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 defined by the vehicle manufacturers.

The SAE J1979 standard currently defines ten 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
0A - request permanent trouble codes

Vehicles are not required to support all of the 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 of them). Within each mode, PID 00 is reserved to show which PIDs are supported by that mode. Mode 01, PID 00 must be supported by all vehicles, and can be accessed as follows…

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. If you have been experimenting, the state of your interface may be unknown, so reset it by sending:

```
>AT Z
```

You will see the interface leds flash, and then the IC should respond with 'ELM327 v2.2', followed by a prompt character. Now, you may choose a protocol that the ELM327 should connect with, but it is usually easier to simply select protocol '0' which tells the IC to search for one:

```
>AT SP 0
```

That's all that you need to do to prepare the ELM327 for communicating with a vehicle. At the prompt, issue the mode 01 PID 00 command:

```
>01 00
```

The ELM327 should say that it is 'SEARCHING...' for a protocol, then it should print a series of numbers, similar to these:

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

The 41 in the above signifies a response from a mode 01 request (01 + 40 = 41), while the second number (00) 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 of 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
```

If the engine is running, the response might be:

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

The returned value (1A F8) is actually a two byte hex number that must be converted to a decimal value to be useful. Converting it, we get a value of 6904, which seems like a very high value for engine rpm.

Talking to the Vehicle (continued)

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. A value of 1726 rpm is much more reasonable.

Note that these examples asked the vehicle for information without regard for the type of OBD protocol that the vehicle uses. This is because the ELM327 takes care of all of the data formatting and translation for you. Unless you are going to do more advanced functions, there is really no need to know what the protocol is.

The above examples showed only a single line of response for each request, but the replies often consist of several separate messages, either from multiple ECUs responding, or from one ECU providing messages that need to be combined to form one response (see 'Multiline Responses' on page 43). In order to be adaptable to this variable number of responses, the ELM327 normally waits to see if any more are coming. If no response arrives within a certain time, it assumes that the ECU is finished. This same timer is also used when waiting for the first response, and if that never arrives, causes 'NO DATA' to be printed.

There is a way to speed up the retrieval of information, if you know how many responses will be sent. By telling the ELM327 how many lines of data to receive, it knows when it is finished, so does not have to go through the final timeout, waiting for data that is not coming. Simply add a single hex digit after the OBD request bytes - the value of the digit providing the maximum number of responses to obtain, and the ELM327 does the rest. For example, if you know that there is only one response coming for the engine temperature request that was previously discussed, you can send:

```
>01 05 1
```

and the ELM327 will return immediately after obtaining only one response. This may save a considerable amount of time, as the default time for the AT ST timer is 200 msec. (The ELM327 still sets the timer after sending the request, but that is only in case the single response does not arrive.)

Some protocols (like J1850 PWM) require an acknowledgement from the ELM327 for every message sent. If you provide a number for the responses that is too small, the ELM327 will return to the prompt too early, and you may cause bus congestion while the ECU tries several times to resend the messages that were not acknowledged. For this reason, you must know how many responses to expect before using this feature.

As an example, consider a request for the vehicle identification number (VIN). This number is 17 digits long, and typically takes 5 lines of data to be represented. It is obtained with mode 09, PID 02, and should be requested with:

```
>09 02
```

or with:

```
>09 02 5
```

if you know that there are five lines of data coming. If you should mistakenly send 09 02 1, you might cause problems.

This ability to specify the number of responses was really added with the programmer in mind. An interface routine can determine how many responses to expect for a specific request, and then store that information for use with subsequent requests. That number can then be added to the requests and the response time can be optimized. For an individual trying to obtain a few trouble codes, the savings are not really worth the trouble, and it's easiest to just make a request, without regard to how many responses are expected.

We offer one additional warning when trying to optimize the speed at which you obtain information from vehicles. Prior to the APR2002 release of the J1979 standard, sending J1850 requests more frequently than every 100 msec was forbidden. With the APR2002 update, scan tools were allowed to send the next request without delay if it was determined that all the responses to the previous request had been received. Vehicles made prior to this time may not be able to tolerate requests at too fast a rate, so use caution with them.

Hopefully this has shown how typical requests are made using the ELM327. If you are looking for more information on modes and PIDs, it is available from the SAE (www.sae.org), from ISO (www.iso.org), or from various other sources on the web.

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 generally not until a request needs to be sent (but you can force one with the FI and SI commands). 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 the 'ON' position 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 as 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, as the default settings work well with most systems.

Periodic (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 connection 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 that these are being sent, simply watch the OBD transmit LED – you will see the periodic 'blips' as the ELM327 sends each one.

The ELM327 normally sends wakeup messages after 3 seconds of no activity (this time is adjustable with the AT SW command). This is within the five second limit set by the standard.

The default content of these messages vary depending on the protocol - for ISO 9141, the ELM327 will send 68 6A F1 01 00, and it sends C1 33 F1 3E for ISO 14230 (KWP). If you would prefer that a different message be sent, simply use the Wakeup Message command to define it.

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 11 22 33 44 55 (with a checksum byte following). You do not provide the checksum byte - it is 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 allows for messages of one to six bytes total, not including the checksum.

The ELM327 does not currently support periodic messages for the CAN protocols. If you require that function, please consider using our ELM329 CAN Interpreter product.

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 J1979 document 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 more 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
- not erase permanent (mode 0A) trouble codes (these are reset by the ECU only)

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 must not be running) before they will respond to a mode 04 command.

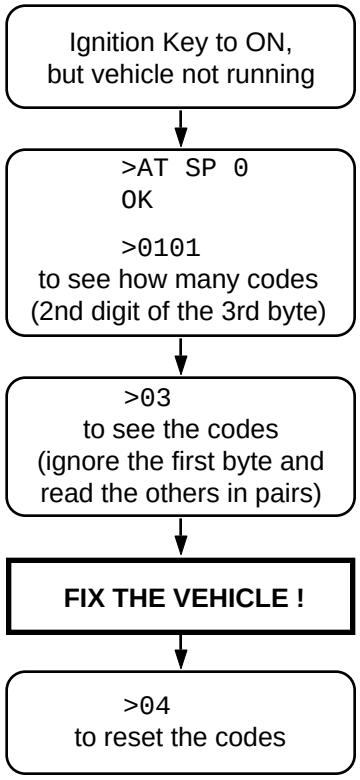
That is all there is to clearing trouble codes. Once again, do not accidentally send the 04 code!

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:



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 a protocol 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' before the protocol number, as follows:

```
>AT SP A2
OK

>AT DP
AUTO, SAE J1850 VPW
```

Now, the ELM327 will try protocol 2, but will then automatically begin searching for another protocol should the attempt to connect with protocol 2 fail (as would happen when you try to connect to your 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 should send the ELM327 an AT SS command first, or step through each protocol with the TP command.

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 then attempt to connect using your headers and your data, and only if that fails (and you have chosen the protocol with AT SP An) 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 using the 'Auto' option when searching (you may need to send AT SP 0) works very well. After the initial search, the protocol used by your vehicle becomes the new default, but it is still able to search for another, without your having to say AT SP 0 again.

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 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 occurred. 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

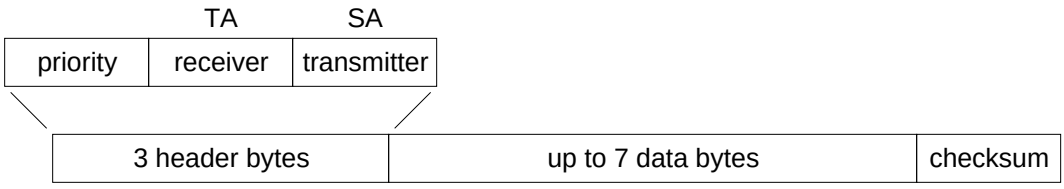


Figure 3. An OBD Message

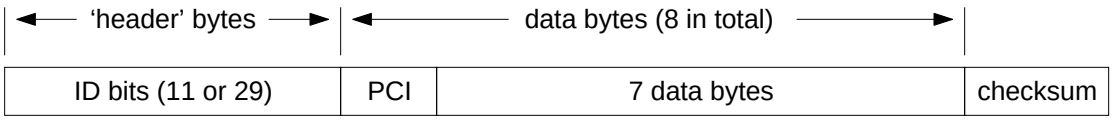


Figure 4. A CAN OBD Message

OBD Message Formats (continued)

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), 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 either 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 with advanced messages such as those for physical addressing. The next section provides a discussion on how to do this…

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 echoes 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

Setting the Headers (continued)

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
```

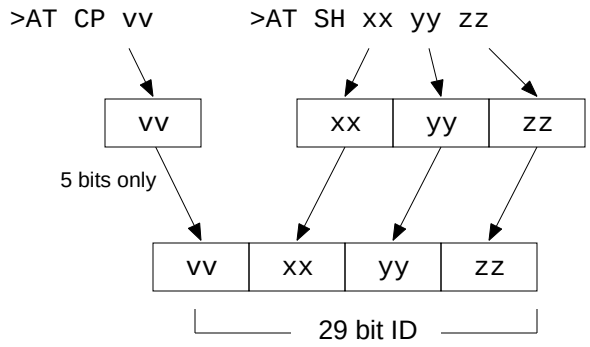
From this point on, all messages that the ELM327 sends will use these three bytes for the header. All that needs to be done now is to request data from the vehicle. For physical addressing, this is often done using mode 22:

```
>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 (00 00 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 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 the SAE standards 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. This is how they 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 (it is the default value used by the ELM327). Since it is rarely changed, it can be assigned separately from the other header bytes, using the CP command.

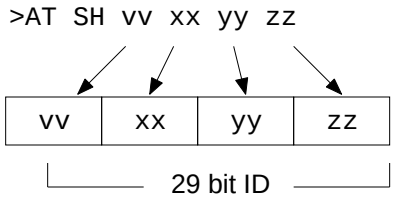
The next byte ('xx') describes the type of message that this is, and is set to hex DB for functional

Setting the Headers (continued)

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 similar (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. Pages 54 and 55 provide more details of the J1939 message structure.

Another method to define all 29 CAN ID bits at once is with the four byte version of the SH command. Simply provide all 8 nibbles in one command:



As with the AT CP command, only 5 bits are used from the first byte (the 3 most significant bits are ignored).

The final header format to discuss is that used by 11 bit CAN systems. They also use a priority and address structure, but shorten it into roughly three nibbles rather than three bytes. The ELM327 uses the same storage locations for these values, so you may use the methods just discussed to assign 11 bit values (but only the least significant bits are used for any messages - the others are ignored).

11 bit CAN 'headers' are normally assigned using a special 'short' version of the AT SH command that uses only three hex digits. Since three digits are actually 12 bits, the most significant bit is ignored, and can have any value. As an example of this instruction, to assign an ID of 7DF, simply send:

```
>AT SH 7DF
OK
```

The OK shows that the ELM327 has accepted your value.

The 11 bit ISO15765-4 CAN standard defines both functional addressing (ID/header = 7DF) and physical addressing (7En) for making requests. Generally, you do not know physical addresses at first, but you do know (from the standard) that the OBD functional address is 7DF.

In order to learn the physical addresses, turn the headers on, and watch what address the replies are from, 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 00
7E8 06 41 00 BE 3F B8 13 00

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

This has just been a quick overview on how to change headers (and ID bits). Hopefully it has been enough to at least help to get you started. We do not recommend experimenting too much without a copy of the standards though, as some aspects are extremely difficult to figure out without them.

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 the receiver must assemble into one complete message.

One example of this is a request for the 17 digit vehicle identification number, or VIN. This is available from newer vehicles using a mode 09, PID 02 request (but was not initially a requirement, so many older vehicles do not support it). Here is one example of a response that might be obtained from a 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
```

The first two bytes (49 and 02) on each line of the response are used to show that the information is in reply to an 09 02 request. The next byte shows which response it is, while the remaining four bytes are the data bytes that are being sent. Assembling the data in the order specified by the third byte, and ignoring the first few 00's (they are filler bytes - see J1979) gives:

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

The data values actually represent the ASCII codes for the digits of the VIN. Using an ASCII table to convert these into characters gives the following VIN 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
```

The 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. CAN systems add this single hex digit (it goes from 0 to F then repeats), to aid in reassembling the data, instead of the byte value that the J1850 vehicle did.

The first line of this response says that there are 014 bytes of information in total. That is 14 in hex, or 20 in decimal terms, which agrees with the 6 + 7 + 7 bytes shown on the three lines. The VIN numbers are generally 17 digits long, however, so how do we assemble the number from 20 digits?

This is done by discarding the first three bytes of the message. The first two are the familiar 49 02, as this is a response to an 09 02 request, so are not part of the VIN. The third byte (the '01'), tells the number of data items that are to follow (the vehicle can only have one VIN, and this agrees with that). Ignoring the third byte 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.

From these two examples, you can see that the format of the data received may depend on the protocol used to transmit it. For this reason, a copy of the SAE J1979 standard would be essential if you are planning to do a lot of work with this, for example if you were writing software to display the received data.

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. First, 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 to the 01 00 request. With modern vehicles, 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 number of data bytes do not agree with the response), and the following group is very confusing, as it has two segment twos. It seems that two ECUs are responding and the information is getting mixed up. Which ECU do the responses belong to? The only way to know 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, which are the CAN IDs representing ECU#1 and ECU#2, respectively. Grouping the responses by ECU 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, but hopefully this has helped you with some of the more common ones. If you are still having trouble, we urge you to purchase the relevant standard, and study it.

CAN Message Types

The ISO 15765-4 (CAN) standard defines several message types that are to be used with diagnostic systems. Currently, there are four 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 in addition to what is known as a PCI (Protocol Control Information) byte. The PCI byte is always the first of the data bytes, 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 see it in the responses.)

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 multi-frame message is about to be sent, and tells the receiver just how many data bytes to expect. The length descriptor is limited to 12 bits, so a maximum of 4095 byes can be received at once using this method.

Consecutive Frame messages are sent after the First Frame message to provide the remainder of the data. Each Consecutive Frame message includes a single hex digit ‘sequence number’ that is used to 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 that 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 ‘Altering Flow Control Messages’ section, on page 61.

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’.

Multiple PID Requests

The SAE J1979 (ISO 15031-5) standard allows requesting multiple PIDs with one message, but only if you connect to the vehicle with CAN (ISO 15765-4). Up to six parameters may be requested at once, and the reply is one message that contains all of the responses.

For example, let us say that you need to know engine load (04), engine coolant temperature (05), manifold pressure (0B), and engine rpm (0C) on a regular basis. You could send four separate requests for them (01 04, then 01 05, then 01 0B, etc.) or you could put them all into one message like this:

```
>01 04 05 0B 0C
```

to which, a typical reply might be:

```
00A
0: 41 04 3F 05 44 0B
1: 21 0C 17 B8 00 00 00
```

The reply is a multiline one, as was discussed in a previous section. Looking at the reply in detail, the first line tells us that it is 00A (decimal 10) bytes long, so we only pay attention to the first ten bytes of the following lines (and ignore the final three 00's on the last line). The first byte is 41, which tells us that the message is a response to an 01 request.

Following the 41 is the actual information, with the PID numbers followed by their data bytes. You will need to know how many data bytes to expect in order to make sense of it in most cases.

The order in which you ask for the PIDs should not matter. For example, the previous request might have been sent as:

```
>01 0B 04 0C 05
00A
0: 41 0B 21 04 3F 0C
1: 17 B8 05 44 00 00 00
```

in which case, the responses might be as shown above (but the order in which the PIDs appear in the response does not have to match the order in which they were requested).

Using this technique, you can make more efficient use of the data bus. The cost is the extra work that you must do in creating the requests, and in parsing each response. If you are writing software to do this, the time initially taken may well be worth it, but if you are typing requests at a terminal screen, it is very unlikely that this will be of benefit to you.

Response Pending Messages

Until the ELM327 v2.1 update, the ELM327 never looked at the content of the OBD messages that it handled. From version 2.1 on, it now looks at them however, specifically looking for 'response pending' replies.

'Response Pending' messages are special ones sent by KWP (ISO 14230) , or CAN (ISO 15765) ECUs to say "Wait, I'm busy." When the scan tool receives one of these replies, the J1979 standard states that it should continue waiting for up to 5 seconds more for the requested information to arrive. Further, if more 'Response Pending' messages arrive, the scan tool should extend the wait period by another 5 seconds each time.

The Response Pending message will always be of the form:

```
7F xx 78
```

where the xx represents the Mode (or SID) that was being requested. There is no feedback as to the PID requested. As ECUs get busier, these messages are becoming more common, so we decided to provide support for them in the ELM327.

If bit 2 of PP 2A is set (it is by default), the ELM327 will automatically change the (AT ST) timeout period to 5 seconds for you if it sees a Response Pending message. This is only for the CAN and KWP (ISO 14230) protocols as per the standard. The CAN protocol support is not normally limited to only ISO 15765, but may be if you set PP 2A b0 to '1'.

Note that the current implementation of this feature does not keep track of multiple ECUs, some of which may reply immediately, and some that may reply with response pending messages. For this reason, there may conceivably be circumstances when you may need to filter for only one ECU address when receiving a Response Pending reply.

CAN Receive Filtering - the CRA Command

When receiving CAN data, the ELM327 actually retrieves every message from the CAN bus, and then decides whether or not to display it, based on criteria that you or the firmware set. The 'CAN Receive Address' or CRA command may be used to define this criteria for you, in one simple step.

As an example, assume that the only messages that you wish to see are those that have the CAN ID 7E9. To see only them, tell the ELM327 that the receive address should be 7E9:

```
>AT CRA 7E9
```

and the ELM327 will set the necessary values so that the only messages that are accepted are the ones with ID 7E9.

If you do not want an exact address, but would prefer to see a range of values, for example all the OBD addresses (those that begin with 7E), then simply use an 'X' for the digit that you do not want the ELM327 to be specific about. That is, to see all messages with CAN IDs that start with 7E (7E0, 7E1, 7E2,... 7EE, and 7EF), send:

```
>AT CRA 7EX
```

and the ELM327 will set the necessary values for you.

This command works exactly the same way for the 29 bit IDs. For example, if you wish to see all messages that are being sent from the engine ECU (address 10) to the scan tool (address F1), then you can send:

```
>AT CRA XX XX F1 10
```

and all the settings will be taken care of for you.

If you wish to be more specific and see only the OBD replies sent by the engine to the scan tool, you would say:

```
>AT CRA 18 DA F1 10
```

and again, the ELM327 makes the necessary changes for you.

Perhaps you do not care which device is sending the information, but you want to see all messages that start with 18 DA and are being sent to the scan tool. For this, use the character 'X' to tell the ELM327 that you do not care what value a digit has:

```
>AT CRA 18 DA F1 XX
```

and the ELM327 takes care of the details for you.

When working with J1939 data, the ELM327 normally formats the data for you, in order to separate the priority from the PGN information. This is usually not a concern when using the CRA command, except when you are trying to filter for a specific priority. For example, you might typically see:

```
>AT MA
3 0FE6C 00 FF FF FF FF FF FF 40 B5
6 0FEEE 00 15 50 FF FF FF FF FF FF
6 0FEF5 00 FE FF FF FF 19 00 23...
```

The single priority digit out front (the 3 or 6 above) as well as the leading 0 with the PGN information are actually part of the first two digits (5 bits) of the ID, and need to be interpreted as such, in order to use the CRA command. It may be easier if you turn off the J1939 header formatting in order to see this:

```
>AT JHF0
OK

>AT MA
0C FE 6C 00 FF FF FF FF FF FF 40 B5
18 FE EE 00 15 50 FF FF FF FF FF FF
18 FE F5 00 FE FF FF FF 19 00 23...
```

This more clearly shows the four bytes that need to be defined for the CRA command to be set. To search for all messages that begin with 6 0FEF5, you would actually need to send the command:

```
>AT CRA 18 FE F5 XX
```

In summary then, the CRA command allows you to tell the ELM327 what ID codes to look for, and the letter 'X' may be used in it to represent any single digit that you do not want the ELM327 to be specific about. This is usually selective enough for most applications, but occasionally, there is a need to be specific down to the bit level, rather than to the nibble. For those applications, you will need to program a separate mask and filter, as we show in the next section.

Using the CAN Mask and Filter

Filtering of CAN messages (that is, deciding which to keep and which to reject), is usually handled most easily with the CRA command. The CRA command only allows for definition to the nibble level, however - if you need more selectivity (to the bit level), you must program the mask and filter.

Internally, the ELM327 configures an 'acceptance filter' with 1's and 0's based on the type of message that it wishes to receive (OBD, J1939, etc.). This pattern is then compared to the ID bits of all incoming messages. If the two patterns match, then the entire message is accepted, and if they do not, the message is rejected.

Having to match all 11 or 29 bits of the ID may be very restrictive in some cases (and would require a very large number of filters for some applications). To allow a little more flexibility in what to accept, and what to reject, a mask is also defined, in addition to the filter. This mask acts just like the type worn on your face - some features are exposed and some are hidden. If the mask has a '1' in a bit position, that bit in the filter must match with the bit in the ID, or the message will be rejected. If the mask bit is a '0', then the ELM327 does not care if that filter bit matches with the message ID bit or not.

As an example, consider the standard response to an 11 bit OBD request. ISO15765-4 states that all responses will use IDs in the range from 7E8 to 7EF. That is:

1. There must always be a '7 ' (binary 111) as the first nibble (so the filter should have the value 111 or 7). All 3 bits are relevant (so the mask should be binary 111 or 7). Note that this first nibble is only 3 bits wide for the 11 bit CAN ID.

2. There must always be an 'E' (binary 1110) in the second position, so the filter needs to be of value 1110 or E. Since all 4 bits are relevant, the mask needs to be of value 1111 or F.

3. If you analyze the patterns for the binary numbers from 8 to F, you will see that the only thing in common is that the most significant bit is always set. That is, the mask will have a value of 1000 since only that one bit is relevant, and you do not care what the other bits are. The filter needs to be assigned a value that has a 1 in the first position, but we do not care what is in the other three positions. In the following, we use 0's in the positions (but it doesn't really matter).

Putting this together, the filter will have a value:

```
111 1110 1000 = 7E8
```

and the mask will have a value:

```
111 1111 1000 = 7F8
```

In order to make these active, you will need to issue both a CAN Filter and a CAN Mask command:

```
>AT CF 7E8
OK

>AT CM 7F8
OK
```

From that point on, only the IDs from 7E8 to 7EF will be accepted by the chip.

The 29 bit IDs work in exactly the same way. For example, assume that you wish to receive only messages of the form:

```
18 DA F1 XX
```

where XX is the address of the ECU that is sending the message, but you do not care what the value is (this is the standard OBD response format). Putting 0's in the mask for don't care bits, then the mask needs to be set as follows:

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

(as every bit except those in the last byte are relevant) while the filter may be set to:

```
>AT CF 18 DA F1 00
OK
```

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' replies. If this happens, and you are unsure of why, you may want to reset everything to the default values (with AT CRA, AT D, or possibly AT WS) and start over.

Quite likely, you will never have to use the CM and CF commands. If you do, then creating your own masks and filters can be difficult. You may find it helpful to draw the bit patterns first, and think about which ones matter, and which ones do not. It may also help to connect to a vehicle, apply test settings, and send AT MA to see how the settings affect the displayed data.

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 by default, the ELM327 remains silent while monitoring, and does not respond with any signals. This means that no periodic 'wakeup' messages are sent, no IFRs are sent, and the CAN module does not generate any message acknowledge bits.

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 any character that you send will be discarded, and will have no effect on any subsequent commands.

The time it takes to respond to such an interrupt 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 printing 'STOPPED' and returning to wait for your input, so it is best to wait for the prompt character ('>') to be sent, or the Busy line to go low, before beginning to send a new 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 may 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, you only need to 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 so all messages that begin with '0' as the first digit will be displayed.

It is best not to use the AT MT or MR commands when monitoring CAN systems. The ELM327 provides another command (AT CRA) that allows much better control over the received data - see the next section for complete details.

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 are 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 (which AT Z does), 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 of the 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. Since some of the Programmable Parameters are only read during a system reset, you may have to follow this command with a reset command:

```
>AT Z
```

after which, you can 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 section on 'Programmable Parameters' (pages 69 and 70) for more details.

Using Higher RS232 Baud Rates

A serial interface has always been provided with our ELM OBD products, largely due to its versatility. Older computers, microprocessors and PDAs can all interface easily with it, as can USB, Bluetooth, ethernet and WiFi modules. It is simply one of the most versatile interfaces available.

Originally, our users almost exclusively used the traditional RS232 interface to connect our integrated circuits to their computers. Interface circuits were easily made, or purchased, and could be used with a very large range of devices. The large voltage swings and cable capacitance worked against using the interface for very high data rates, however, so we set the default data rate of the ELM327 to a conservative 38.4 kbps.

If your application needs a traditional RS232 interface, then by all means use one. We offer a few suggestions in the Example Applications section that you might try. The discrete version that we show works extremely well at speeds of up to 57600 bps, and depending on several factors, it may also work well at speeds as high as 115200 bps.

If you would like to operate your interface at speeds of 115200 bps or higher, there are several single IC solutions that are available. These include devices such as the ADM232A from Analog Devices (www.analog.com), or the popular MAX232 series of integrated circuits from Maxim Integrated Products (www.maximintegrated.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 devices are only rated for operation up to 120 kbps, however, so may not be suitable for very high data rates - be sure to check the manufacturers data sheet before committing to a design.

An RS232 interface is typically limited to operating at speeds of about 230.4 kbps maximum. If you wish to go higher than that, then you must consider alternatives - one of which is to use USB.

Almost all computers that are made today have replaced the once familiar RS232 port with a USB one. Software is readily available to make these look like traditional serial ('COM') ports, and modules are available for connecting to circuits like the ELM327. Several manufacturers offer these modules (often called 'bridge' circuits) – try the CP2102 from Silicon Labs (www.silabs.com) or the FT232R from Future Technology Devices (www.ftdichip.com). If planning to use higher baud rates, 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 77 for more details.

The default configuration for the ELM327 typically provides a data rate of 38400 baud. There are two ways that this rate can be changed – either permanently with a Programmable Parameter, or temporarily with an AT command.

Programmable Parameter '0C' is the ELM327 device setting that stores the baud rate divisor. 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, your stored rate will become the new data rate after the next reset. As an example, perhaps you would like to have the ELM327 use a baud rate of 57.6 kbps, rather than the factory setting of 38.4 kbps. To do this, you need to 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 (the baud rate in kbps is 4000 divided by the value of PP 0C). To obtain a setting of 57.6 then, 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 the set value command:

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

then enable the new PP 0C value for use:

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

from that point on, 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 baud rate with PP 0C. Software developers will not usually 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. The chart shown here tries to explain just how this command should be used.

Using Higher RS232 Baud Rates (continued)

The sequence begins with the PC making a request for a new baud rate divisor, with the BRD command. For example, to try the 57.6K rate that was previously discussed, the controlling PC would send:

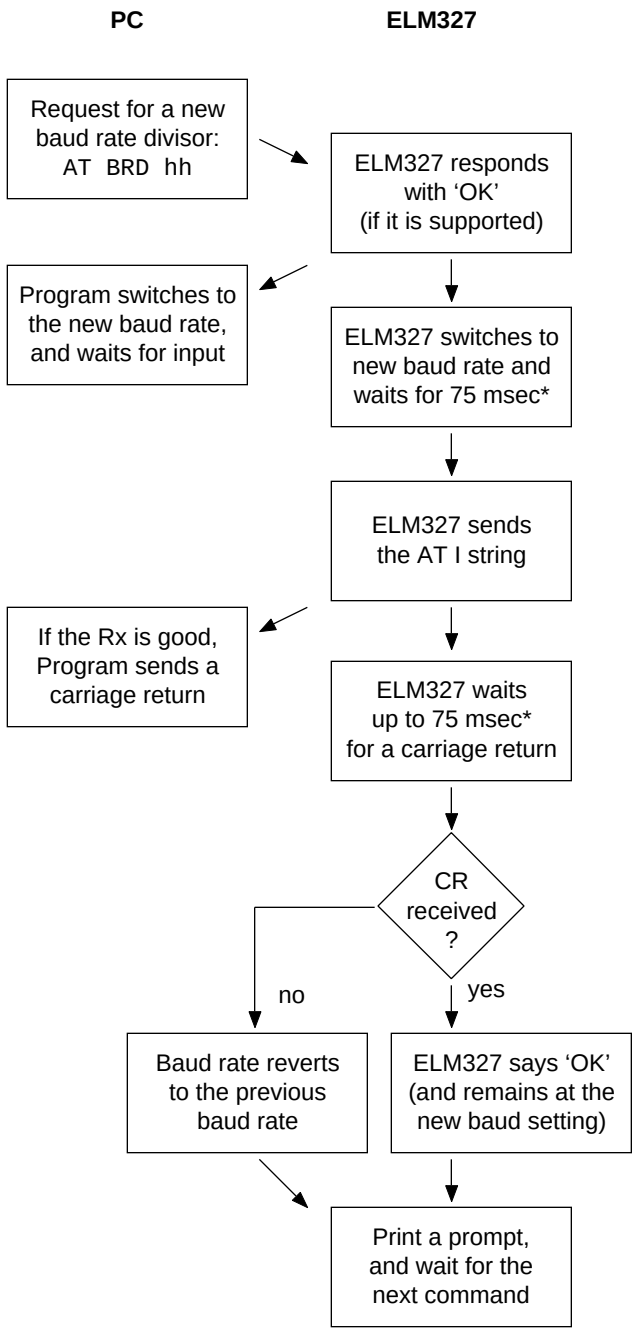
```
AT BRD 45
```

If the ELM327 firmware is a very old version, it will not support this command and will return with the familiar '?' response. 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 the 'OK', the ELM327 then switches to the new (proposed) baud rate, and then simply waits the time set by the BRT command (it is 75 msec by default). 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 v2.2') to the PC at the new baud rate (followed by a carriage return character and optionally, a linefeed character) 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 see a carriage return in the 75 msec 'window', the baud rate will revert back to the original 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.

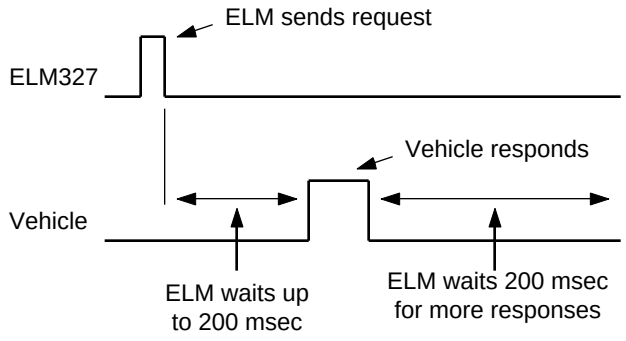


* the 75 msec time is adjustable with the AT BRT hh command

Setting Timeouts - AT ST and AT AT Commands

Users often ask about how to obtain faster OBD scanning rates. There is nothing that we can do about how fast (or slow) a vehicle is to respond, but we can optimize how the ELM327 handles the responses.

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



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

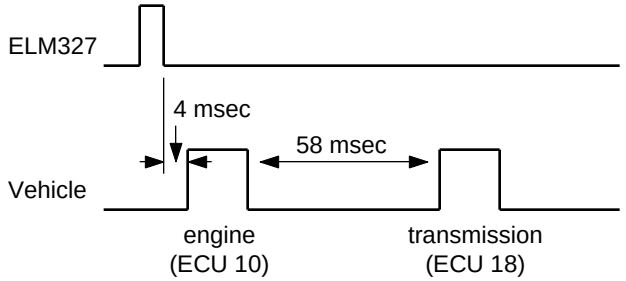
After each reply has been received, the ELM327 must wait to see if any more replies are coming (and it uses the same internal timer to stop the waiting if no more replies arrive). With our initial OBD products (the ELM320, ELM322 and ELM323) we found that older vehicles often needed a timeout setting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

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



The engine controller 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 90 msec. With a timeout of 90 msec, the maximum scan rate would be about 6 readings per second.

Surely there has to be a way to eliminate that final timeout, if you know how many responses to expect? There is a way – instead of sending 01 00 for the above request, the ELM327 will 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 knows the number of responses to expect. This one change might give you 10 to 12 responses per second, instead of the 6 obtained previously.

We do caution you to use this 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 replies, looking for a response. This will lead to unnecessary network congestion, which must 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.

SAE J1939 Messages

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

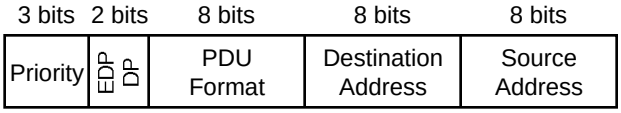
The following will discuss a little of how data is transferred using the J1939 standard. Considerably more information is provided in the Society of Automotive Engineers (SAE) standards documents, so if you are going to be doing a lot of work with J1939, it may be wise to purchase copies of them. At minimum, the J1939-73 diagnostics, the J1939-21 data transfer, and the J1939-71 vehicle application documents should be purchased. Another great reference for this work is the HS-J1939 two book set, also available from the SAE.

The current version of the J1939 standard allows only one data rate (250 kbps), but work is underway to amend the standard so that an alternate rate of 500 kbps will also be allowed. For the purpose of this discussion, the data rate is not important - it is the format of the information that we will discuss.

All CAN messages are sent in ‘frames’, which are data structures that have ID bits and data bytes, as well as checksums and other items. Many of the J1939 frames are sent with eight data bytes, although there is no requirement to do so (unlike ISO 15765, which must always send eight data bytes in each frame). If a J1939 message is eight bytes or less, it will be sent in one frame, while longer messages are sent using multiple frames, just like ISO 15765. When sending multiple frames, a single data byte is used to assign a ‘sequence number’, which helps in determining if a frame is missing, as well as in the reassembly of the received message. Sequence numbers always start with 01, so there is a maximum of 255 frames in a message, or 1785 bytes.

One major feature of the J1939 standard is its very orderly, well defined data structures. Related data is defined and specified in what are called ‘parameter groups’. Each parameter group is assigned a ‘parameter group number’, or PGN, that uniquely defines that packet of information. Often, the parameter groups consist of eight bytes of data (which is convenient for CAN messages), but they are not restricted to this. Many of the PGNs, and the data within them (the SPNs) are defined in the J1939-71 document, and manufacturers also have the ability to define their own proprietary PGNs.

The ID portion of a J1939 CAN frame is always 29 bits in length. It provides information as to the type of message that is being sent, the priority of the message, the device address that is sending it, and the intended recipient. Information within the ID bits is divided roughly into byte size pieces as follows:



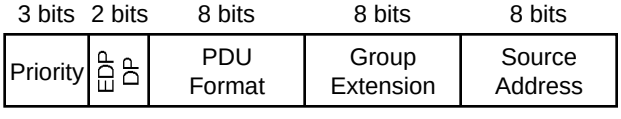
PDU1 Format

The data structure formed by the 29 bit ID, and the associated data bytes is called a Protocol Data Unit, or PDU. When the ID bits have a destination address specified, as is shown above, it is said to be a PDU1 Format message.

The two bits shown between the Priority and the PDU Format are known as the Extended Data Page (EDP), and the Data Page (DP) bits. For J1939, EDP must always be set to ‘0’, while the DP bit is used to extend the range of values that the PDU Format may have. While the DP bit is typically ‘0’ now, that may not be true in the future.

Not all J1939 information is sent to a specific address. In fact, one of the unique features of this standard is that there is a large amount of information that is being continually broadcast over the network, with receivers using it as they see fit. In this way, multiple devices requiring the same information do not have to make multiple requests to obtain it, information is provided at regular time intervals, and bus loading is reduced.

If information is being broadcast over the network to no particular address, then the Destination Address field is not required. The eight bits can be put to better use, possibly by extending the PDU Format field. This is what is done for a PDU2 Format frame, as shown here:



PDU2 Format

So how does one know if they are looking at a PDU1 Format frame that contains an address, or a PDU2 Format frame that does not? The secret lies in

SAE J1939 Messages (continued)

the values assigned to the PDU Format field. If the PDU Format value begins with 'F' (when expressed as a hexadecimal number), it is PDU2. Any other value for the first digit means that it is a PDU1 Format frame, which contains an address.

To summarize, PDU1 format frames are sent to a specific address, and PDU2 frames are sent to all addresses. To further complicate matters, however, PDU1 frames may be sent to all addresses. This is done by sending the message to a special 'global address' which has the value FF. That is, if you see a PDU1 message (where the first digit of the PDU Format byte is not an F), and the Destination Address is FF, then that message is being sent to all devices.

The J1939 recommended practices document provides a list of addresses that should be used by devices. It is particularly important to adhere to this list with the ELM327, as the IC uses a fixed address method and is not able to negotiate a different one, per J1939-81. OBD Service Tools should use either F9 or FA as their address (the ELM327 uses F9). If you wish to change this, you can use the AT TA (tester address) command, or simply define it with the header.

You may use either the AT SH or the AT CP and AT SH commands to assign values for the J1939 ID bits, just the same as with the other CAN protocols. How these are used was discussed previously, but we will repeat it here.

Since the priority (and DP and EDP) values only rarely change, they may be assigned with the CP command. By default, the ELM327 uses a priority of 6 (binary 110), and sets the EDP and DP bits both to 0. The default value for the CP setting is then 110+0+0 (which would be entered as 11000 or 18 in hex). The values for the other bytes of the J1939 ID are entered with the AT SH command, as shown in Figure 5.

This has tried to cover the basics of the J1939 message structure, but if you want more information, you should look at the standards mentioned previously. One other one that gives good examples of actual data is J1939-84 which describes the compliance tests and shows the expected responses.

Even at 250 kbps, J1939 data is transferred at a rate that is more than ten times faster than the previous heavy duty vehicle standard (SAE J1708), and several of the light duty standards. As designers build more into each system, the amount of information required continues to grow, however, so the 500 kbps version of J1939 will be a welcome addition.

Use either: `>AT SH vv xx yy zz`

or: `>AT CP vv` and `>AT SH xx yy zz`

The values will be used as follows:

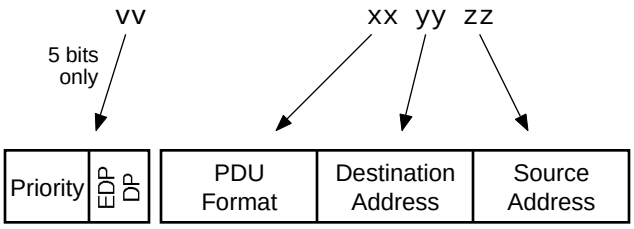


Figure 5. Setting the J1939 CAN ID

Using J1939

This section provides a few examples which show how to monitor an SAE J1939 data bus, and how to make requests of devices that are connected to it.

To begin, you will need to configure the ELM327 for J1939 operation, at the correct baud rate. Protocol A is predefined for J1939 at 250 kbps, which is what most applications require. To use protocol A, send:

```
>AT SP A
```

Protocols B and C may also be used with J1939, if you wish to experiment with other baud rates. To use them for J1939, the option value (in PP 2C or 2E) must be set to 42, and the baud rate divisor (in PP 2D or 2F) must be set to the appropriate value. Perhaps the simplest way to provide an alternate rate is to use the AT PB command, that allows you to set both the options byte (which is always 42), and the baud rate divisor (which is 500k ÷ the desired baud rate) at the same time. For example, to set protocol B for J1939 operation at 500 kbps, simply send:

```
>AT PB 42 01
```

then send:

```
>AT SP B
```

to begin. Note that this setting will not be maintained if the IC is reset, so if you want a more permanent setting, you should store the values in PP 2C and 2D.

J1939 often provides slowly broadcast information, and for this reason, the ELM327 automatically sets a response timeout (AT ST time) as required by the message (typically 1.25 seconds, but it varies). If this is too long for you, or if you are using an older version of our product (ie. v1.3a or older), then you might want to set this time manually (we recommend starting with AT ST FF for the older ICs). You will not do any harm if you set the timeout with a newer chip, but the timing will not be optimal as you will stop the ELM327 from varying the setting based on the type of message being received.

The ELM327 also offers one other variation on the timer setting - the ability to extend the AT ST time by switching a x5 timer multiplier on (see the JTM5 command description). This may be useful or even necessary when requesting data that will have a multiline response, if similar data is already flowing. In these circumstances, there can only be one message like this at a time on the bus, so the response to your request might have to wait while an initial response completes (and this could take more than the normal ST time, since broadcast responses must be spaced at least 50 msec apart). If you know that a reply should be coming, and you are seeing ‘NO DATA’ responses, then send AT JTM5 and try it again, as that may be the problem. Restore the timer multiplier to normal with AT JTM1.

Once the J1939 protocol is selected, and the timeout value has been adjusted, the ELM327 is ready for a command. The first one that we will send is called a DM1 or ‘diagnostic message 1’, which provides the currently active diagnostic trouble codes. DM1 is one of more than 50 predefined diagnostic messages, and is special in that it is the only one that is broadcast continually over the bus at regular intervals. The ELM327 has an AT command that is used to obtain the DM1 trouble codes:

```
>AT DM1
```

If you are connected to a vehicle, you should now see messages printed at one second intervals. If you are only connected to a single device (for example, with a simulator on the bench, or to a device with a single CAN data port), you may see data with <RX ERROR printed beside it. This is because the receipt of the data is not being acknowledged by any device on the bus (certainly not the ELM327, as it is by default a completely silent monitor). See our ‘AN05 - Bench Testing OBD Interfaces' application note for more information on this, and some advice on what to do. If you have a v1.4b or newer chip, you do not have to take special measures, however. Simply turn off the silent monitoring with:

```
>AT CSM 0
```

and there should be no more RX ERRORs. Once you have this sorted out, repeat the request. If all goes well, you should see several replies, similar to this:

```
00 FF 00 00 00 00 FF FF
00 FF 00 00 00 00 FF FF
```

You will likely need to stop the flow of data by pressing any key on the keyboard. This is because the DM1 command is actually a special form of a monitoring command, and all monitoring needs to be stopped by the user. The response means that there are currently no active trouble codes, by the way.

Using J1939 (continued)

To see the exact same response, you can also Monitor for PGN 00FECA (which is the code for DM1):

```
>AT MP 00FECA
```

Note that the ELM327 requires that you send hex digits for all data, as shown above (and as used by all other protocols). Many of the PGN numbers are listed in the J1939 standard as both a decimal and a hex number, so choose the hex version.

You will likely find in your testing that the PGNs you encounter often begin with a 00 byte as above. To simplify matters for you, the ELM327 has a special version of the MP command that will accept a four digit PGN, and assumes that the missing byte should be 00. An equivalent way to ask for 00FECA is then:

```
>AT MP FECA
```

which is a little more convenient.

One feature of the ELM327 is the ability to tell the IC how many messages to retrieve when monitoring for PGNs. For example, to see only two responses to the MP FECA command, send:

```
>AT MP FECA 2
```

This saves having to send a character to stop the flow of data, and also is very convenient when dealing with multiline messages. While the standard OBD requests allow you to define how many frames (ie lines) of information are to be printed with a similar single digit, the single digit with the MP command actually defines how many complete messages to obtain. For example, if the DM1 message is 33 lines long, then sending AT MP FECA 1 will cause the ELM327 to show all 33 lines, then stop monitoring and print a prompt character.

By default, all J1939 messages have the 'header' information hidden from view. In order to see this information (actually the ID bits), you will need to turn the header display on:

```
>AT H1
```

A single response to FECA might then look like:

```
>AT MP FECA 1
6 0FECA 00 00 FF 00 00 00 00 FF FF
```

Notice that the ELM327 separates the priority bits from the PGN information. The ELM327 also uses only one digit to represent the two extra PGN bits, both of which may seem unusual if you are used to different software. We find this a convenient way to show the actual J1939 information in the header.

If you prefer to see the ID bits separated into bytes instead, simply turn off the J1939 header formatting with:

```
>AT JHF0
```

Repeating the above request would then result in a response of this type:

```
>AT MP FECA 1
18 FE CA 00 00 FF 00 00 00 00 FF FF
```

The differences are clearly seen. If displaying the information in this manner, remember that the first 'byte' shown actually represents five bits, and of them, the leftmost three are the priority bits.

The MP command is very useful for getting information in a J1939 system, but not all information is broadcast. Some information must be obtained by making a query for it. Just like the other OBD requests where you specify the information that you need (with a mode and a PID), to make a query in a J1939 system, you provide the PGN number and the system responds with the required data.

For example, to request the current value of the engine coolant temperature (which is part of PGN 00FEEE), you would send a request for PGN 00FEEE, and extract the data. To do this, send:

```
>00FEEE
```

to which you might receive:

```
6 0FEEE 00 8C FF FF FF FF FF FF FF
```

if the headers were on. Note that if you request a PGN that is already being broadcast, you may very well receive many replies, as the ELM327 configures itself to receive anything that is related to the PGN requested.

If you are familiar with the J1939 standard, you will be aware that it actually specifies a reverse order for the sending of the data bytes of a PGN request. That is, the data bytes for the above request are actually sent as EE FE 00, and not as 00 FE EE. Since it can be very confusing to have to reverse some numbers and not others, the ELM327 automatically handles this for you, reversing the bytes provided. In this way, you

Using J1939 (continued)

can directly request PGNs using numbers as they are written on the page (if they are written as hex digits), and the ELM327 will make it work for you. If you do not want the ELM327 to alter the byte order, the feature can be disabled (by sending an AT JS command).

The ELM327 always assumes that when you start making requests of this type, you do not know what devices are connected to the J1939 bus. That is, by default the ELM327 sends all requests to the 'global address' (ie all devices), and then looks for replies. Often, this works well, but J1939 devices are not required to respond to such general inquiries, and may not if they are busy. For this reason, it is usually better to direct your queries to a specific address, once it is known.

In order to determine the address to send to, you may have to monitor the information on the bus for a while. Make sure that the headers (ID bits) are being displayed, and note what is shown in the Source Address position, which is immediately before the data bytes. In the previous example, this would be 00 (which J1939 defines as the address for engine #1). As an example, let us assume that it is engine #1 that you wish to direct your queries to. To do this, you will want to change the Destination Address from FF (the global address) to 00 (engine #1).

By default, the ELM327 uses 6 0EAFF F9 for the ID bits of all requests (or 18 EA FF F9 if you prefer). That is, it uses a priority of 6, to make a request (EA) to the global address (FF) by the device at F9 (the scan tool). An EA request is often referred to as a request using PGN 59904 as the address EA00 in hex is 59904 in decimal.

Since you only wish to alter the EAFF F9 portion of the ID bits and not the priority, you may do this with the three byte 'set header' command:

```
>AT SH EA 00 F9
```

The priority bits rarely need to be changed, but if you do need to change them, it may be done with the CAN Priority (AT CP) command.

After making the above change, all data requests will be directed to address 00 (the engine), so don't forget to change the headers if you wish to again make global requests. Note that the AT SH command allows you to change the source (or tester) address at will, so be careful with this as addresses should really be negotiated using the method described in J1939-81 and you might conceivably choose an address that is already in use. The current version of the ELM327 does not support J1939-81 address negotiation, so can not obtain an address for you.

Once the ELM327 has been configured to send all messages to address 00, repeat the request:

```
>00FEEE
6 0E8FF 00 01 FF FF FF FF EE FE 00
```

This response is of the 'acknowledgement' type (E8), which is being broadcast to all (FF) by the device with address 00. The last three data bytes show the PGN requested, in reverse byte order, so we know this is a response to our request. Looking at the other data bytes, the first is not 00 (which we would expect for a positive acknowledgement), it is 01 which means negative acknowledgement. Since all requests to a specific address must be responded to, the device at address 00 is responding by saying that it is not able to respond. That is, retrieve the information using the MP command.

If the ECU had been able to reply to the request, the format of the response would have been slightly different. For example, if a request for engine run time (PGN 00FEE5) had been made, the response might have been like this:

```
>00FEE5
6 0FEE5 00 80 84 1E 00 FF FF FF FF
```

Notice that the PGN appears in the header for these types of replies, and the data bytes are those defined for the SPNs in the PGN.

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. As an example, a multisegment response to a DM2 request might look like this:

```
>00FECB
012
7 0EBF9 00 01 04 FF 50 00 04 0B 54
7 0EBF9 00 02 00 00 01 5F 05 02 31
7 0EBF9 00 03 6D 05 03 03 FF FF FF
```

if the headers are on, and would appear as:

```
>00FECB
012
01: 04 FF 50 00 04 0B 54
```

Using J1939 (continued)

```
02: 00 00 01 5F 05 02 31
03: 6D 05 03 03 FF FF FF
```

if the headers are off. Note that multiframe messages always send eight bytes of data, and fill in unused byte positions with FFs.

With the headers off, a multiline response looks very similar to the multiline responses for ISO15765-4. The first line shows the total number of bytes in the message, and the other lines show the segment number, then a colon, and the data bytes received. Note that the byte count is a hexadecimal value (ie the '012' shown means that there are 18 bytes of data).

The single line that shows 012 in the above (the total number of data bytes) is actually a special type of response, called a 'Connection Management' or 'TP.CM' message. It has a specific format, but the only bytes that are typically relevant are those that provide the total message size in bytes, so that is all that the ELM327 normally shows. In order to see the other bytes, you must turn CAN Auto Formatting off (AT CAF0), and then repeat the request. Note that this will only show the entire TP.CM message if you have a version 1.4b or newer ELM327.

This has been a brief description of how to use the ELM327 in a typical J1939 environment. If you can monitor for information, make global requests as well as specific ones, and receive single or multiframe responses, then you have the tools necessary to diagnose most vehicle problems.

The FMS Standard

Several European heavy duty truck and bus manufacturers have joined to form an organization for standardizing the way in which information is retrieved from these large vehicles. The result of their work is the FMS (or Fleet Management Systems) Standard, and the Bus-FMS Standard.

The FMS standard is based on a subset of the 250 kbps J1939 protocol, which uses only broadcast messages for the information. In order to not compromise the integrity of the vehicle's CAN bus, the standard also specifies a gateway device to provide separation between (potentially unskilled) users and the critical control information on the vehicle.

The information contained in the FMS messages is defined by PGNs, using the same PGN numbers as for J1939. The difference is that they only define a small subset of those specified by J1939.

To monitor the information provided by an FMS gateway, simply use the AT MP command with the appropriate PGN number. We should caution that some information (VIN, software version, etc.) is only transmitted every 10 seconds, so some patience is required when waiting for the data.

The FMS standard is completely open, and still continues to evolve. Originally there were different documents for Bus and Truck, but now they are combined into one. As of this writing, Version 03 (dated 12.03.2012) is the most recent document. For more information, visit either FMS web site:

FMS Standard:
www.fms-standard.com

Bus FMS Standard:
bus-fms-standard.com

The NMEA 2000 Standard

We are occasionally asked about support for the NMEA 2000 marine standard. Elm Electronics does not provide specific support for this protocol, but our ELM327 integrated circuit is very capable of working with it.

While the physical connectors may look quite a bit different than those used for J1939, the CAN interface and the data format is almost identical to that of the J1939 standard. NMEA 2000 uses a 250 kbps data rate, so the easiest way to get started is to select the ELM327's predefined protocol A. This is done with the set protocol to A command:

```
>AT SP A
```

When you are finished and want to use the ELM327 for standard OBDII protocols, don't forget to send the SP 0 or SP 00 command to reset it.

Many of the PGNs used for NMEA 2000 have values that are greater than 65535, so the DP bit is usually set. To monitor for most PGNs then, you can not use the short version of the MP command. For example, to monitor for the Engine Parameters PGN (127488 or hex 1F200), you can not use:

```
>AT MP 1F200
```

as the ELM327 actually interprets that as a request for PGN 1F20, and get 0 replies. To monitor for PGN 1F200, you must send:

```
>AT MP 01F200
```

If you keep the above in mind, the ELM327 will prove to be a handy tool to use while experimenting with NMEA 2000. It does have a couple of limitations that must be kept in mind, though. As mentioned with J1939, it is not capable of address negotiation. Also, the ELM327 does not currently support the Fast Packet protocol, which may be an issue for some users.

For more information on the NMEA 2000 standard, visit the NMEA web site (http://www.nmea.org).

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 response for you as long as the CAN Flow Control option is enabled (CFC1), which it is by default.

In order to provide complete control over the sending of Flow Control messages and their content, the ELM327 defines several AT FC commands.

The way in which the ELM327 responds to a First Frame message is determined by the Flow Control 'mode', as set by the AT FC SM command. There are currently three modes, as shown in the chart. The default Flow Control mode is number '0', meaning that the ELM327 will do everything for you.

Flow Control mode 1 is provided for those users that want complete control over their Flow Control messages. To use it, you must define both the CAN ID (header) and the data bytes that you require to be sent in the response to a First Frame message. Note that 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
```

and 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). Note that we show 11 bits for the ID in this example, but you may also define 29 bit IDs.

The third mode currently supported allows setting the data bytes which are to be sent, but not the ID bits. The ELM327 sets the ID bits in mode 2 from those which were received in the First Frame message – it does not change them at all. 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
```

At any time while you are experimenting, if you should wish to restore the automatic Flow Control responses, simply set the mode to 0:

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

This will immediately restore the responses to their (ISO 15765-4) default settings.

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.

The following chart summarizes the currently supported flow control 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

Flow Control Modes

Note that the ELM327 will only send Flow Control messages if the protocol's data format is ISO15765-4.

Using CAN Extended Addresses

Some vehicles with CAN interfaces use a data format that is slightly different from what we have described so far. The data packets look very similar, except that the first byte is used for the receiver's (ie target's) address. The remaining seven bytes are used as described previously.

We refer to this type of addressing as 'CAN Extended Addressing', and provide support for it with the CEA and CER commands. Perhaps an example would best describe how to use it.

Here is a portion of a data transfer that was taken from a vehicle. For the moment, ignore the first data bytes on each line and only look at the remaining data bytes (that are outlined in grey):

```
7B0 04 02 10 81 00 00 00 00
7C0 F1 02 50 81 00 00 00 00
7B0 04 02 21 A2 00 00 00 00
7C0 F1 10 16 61 A2 01 02 05
7B0 04 30 FF 00 00 00 00 00
7C0 F1 20 DF 01 00 04 09 01
7C0 F1 21 02 05 DF 01 00 04
7C0 F1 22 09 01 00 04 01 00
```

If you are familiar with the ISO 15765 data format, you will recognize that the data bytes shown inside the box appear to conform to that standard. The rows that begin with 02 are Single Frames, the one that starts with 10 is a First Frame, the one with a 30 is a Flow Control, and the others are Consecutive Frames.

The remaining bytes, shown outside the box, are the standard 11 bit CAN ID, and an extra address byte. The lines that have F1 for the extra address are directed to the scan tool (all scan tools generally use F1 as the default address), and the other lines are being sent to the vehicle (at address 04).

As an example, we will set the ELM327 to handle the above messages. We know that the messages use 11 bit IDs with ISO 15765 formatting, and let us assume a baud rate of 50kbps. The command to configure protocol B for this is:

```
>AT PB 81 0A
```

Next, tell the ELM327 to receive all messages that have an ID of 7C0:

```
>AT CRA 7C0
```

and to send with an ID (header) of 7B0:

```
>AT SH 7B0
```

Notice that there was a flow control message that was sent in this group, but it's not quite the same as the one for OBD systems. For this reason, you'll need to define your own flow control with the following three statements:

```
>AT FC SH 7B0
>AT FC SD 04 30 FF 00
>AT FC SM1
```

The final setup statement that you will need is to tell the ELM327 to send messages to CAN Extended Address 04:

```
>AT CEA 04
```

Note that the default CER receive address is the tester address (F1), so we do not need to send a CER command to define the receive address.

Now everything is configured. Next, tell the IC to use this protocol, and to bypass any initiation (as it is not standard OBD, and would likely fail):

```
>AT SP B
>AT BI
```

That's all. To exactly reproduce the flow of data shown previously, you only need to send the relevant data bytes and the ELM327 will add the rest:

```
>10 81
50 81

>21 A2
016
0: 61 A2 01 02 05
0: DF 01 00 04 09 01
1: 02 05 DF 01 00 04
2: 09 01 00 04 01 00
```

Notice that for some reason, this vehicle has sent two segment 0's, but that just means that it doesn't exactly follow the ISO 15765 protocol. The above shows what the responses would look like with the headers off. If they had been on, the data exchange would look like what we showed on the left.

CAN Input Frequency Matching

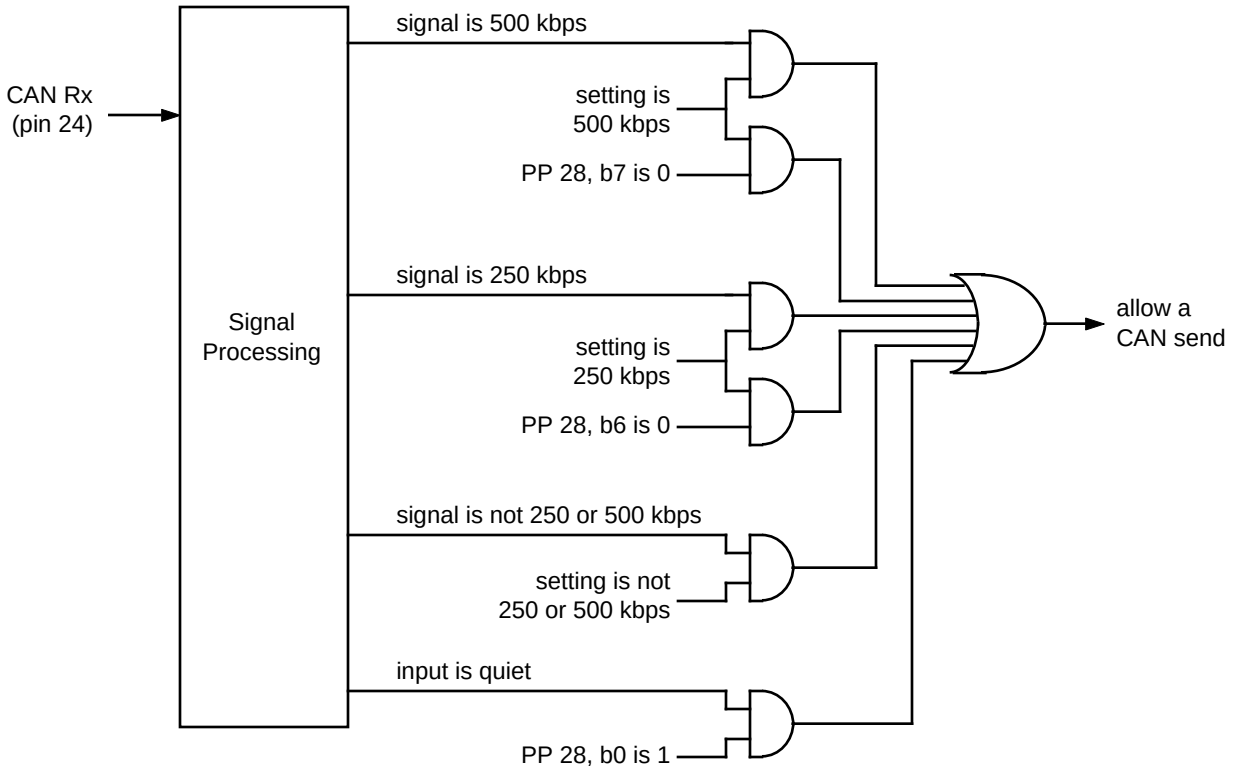
Most modern vehicles have a CAN network connected to pins 6 and 14 of the OBD connector. At one time, however, the use of these pins was left to the vehicle manufacturer, and a number of different systems were connected to them.

In order to prevent the disruption of any connected systems while the ELM327 is searching for a protocol (it sends out requests during a search), the ELM327 now performs several tests on these wires. Prior to firmware version 2.1, the tests simply looked for activity on the wires but were not frequency selective. This meant that, for example, vehicles that had a speedometer signal connected to either pin might be seen as a valid CAN network, and the ELM327 may have sent a request on those wires. The new firmware actually measures the input frequency and requires that it matches that of the selected CAN protocol before any test message can be sent.

The diagram below shows how the logic works. It may seem a little complicated, but what it really says is that for the default settings, a send is allowed if the input signal frequency matches the CAN setting (250 or 500 kbps), or if there appears to be no signal. In addition, if the user is trying a non-standard OBD frequency, but a standard frequency is received, a send will not be allowed.

All bits of PP 28 are set to 1 by default (requiring frequency matching, unless no signal is detected), but may be changed at any time – see the Programmable Parameters section for details.

This logic is only used while searching for a valid protocol. Once a particular protocol is considered to be active, no further frequency checks are made (as it is time consuming). Note that if you should use the AT BI command to bypass the initiation process, this frequency matching test will also bypassed.



Send Logic While Searching for a Protocol

Programming Serial Numbers

A number of our customers have asked for ways to uniquely identify a product that uses our ELM327 integrated circuit. While this is often a request for a means to store a 'serial number', people have also asked for a way to store dates, and version codes, too. The @2 and @3 commands were created to assist with this.

If you send the command AT @2 to a new ELM327 integrated circuit, you will receive an error. That is, you will see a response that looks like this:

```
>AT @2
?
```

In the above dialog, the ELM327 is trying to tell you that either the chip is very old and does not support the @2 command, or that nothing has been programmed into these memory locations yet.

To program characters into the @2 memory, you must provide exactly 12 characters using the AT @3 command. These characters must be in the ASCII printable group, in the range from '!' (hex value 21) to '_' (hex value 5F). Typically, an AT @3 command use would look like:

```
>AT @3 MYBOARD_9906
OK
```

This number can never be altered once it is entered, so you must be sure that you are entering the values properly. If developing code which does this, you may find that purchasing an ELM328 IC will save the expense of trial and error. The ELM328 does not support OBD protocols however, so is not a viable option for other uses.

Once the @3 code is set, it will always be available through the @2 command:

```
>AT @2
MYBOARD_9906
```

That's all there is to using the ELM327 device identifier.

Saving a Data Byte

The ELM327 provides one memory location that can be used to save any single byte of information. This location uses special 'non-volatile' EEPROM memory for storage, so your data is not lost, even if you should turn the power off.

Typically, this memory location is used by the controlling software to store the state of flags that were set by vehicle conditions, by hardware configurations, or by software choices. By storing them in this type of memory, the settings will be remembered between uses of the scan tool.

Storing data is easily done with the Save Data command - for example, to save the value 7F, simply send:

```
>AT SD 7F
OK
```

Data is just as easily retrieved using the Read Data command:

```
>AT RD
7F
```

Since this single byte of data is stored in the internal EEPROM array, it is subject to the usual limits of EEPROM technology - unlimited reads, but typically only about 1 million writes, with a retention time of 40 years (or more). This should not pose any limits to ELM327 users that we are aware of.

The Activity Monitor

The ELM327 contains some firmware that is used to monitor the OBD input pins (ie pins 11, 12, 13, and 24). These routines continually 'poll' the inputs, looking for any active levels on them. Of course, we don't want to look at a pin while we are sending, so there is also some logic to block those inputs during a send, to filter the levels, and also to provide timing. Collectively, these routines are known as the Activity Monitor.

The Activity Monitor software ensures that each OBD input pin is checked at least once every 4 msec (and sometimes as often as every few µsec). This may not be perfect (since a single very short message on an otherwise quiet bus could be missed), but it does make sure that the ELM327 detects normal activity on active busses. The presence or absence of activity can then be used to cause the ELM327 to go to Low Power operation, or wake from it. See the next section for details on that.

If you do not want the Activity Monitor to initiate Low Power operation, you may set PP 0F so that only 'ACT ALERT' reports are provided. Of course, you may also disable that as well, and make your own decisions based on what the Activity Monitor is seeing.

To help with this, the ELM327 offers an instruction that is able to report the current Activity Monitor Counts. The count is an internal value that represents the time since OBD activity was detected. The actual time is given by (AMC value + 1) x 0.65536 seconds. To use it, send the AMC Command and note the value returned:

```
>AT AMC
31
```

In this case, the value returned is 31 hexadecimal (or 49 decimal), which means that the ELM327 has not detected any OBD activity in the past (49+1) x 0.655 = 32.8 seconds. The Activity Monitor Count is limited to a single byte value (max is FF), and internal logic prevents it from 'rolling over'. That is, it will count to FF and stop there.

After a power on or reset, the Activity Monitor is initially disabled (to give you time to perform setup tasks), but it will be enabled as soon as you attempt to send the first OBD message. Also, the monitor is always disabled while the ELM327 is in a monitoring mode so that you can use the ELM327 as a reliable monitor of bus activity (you don't want the IC to go to sleep when it should be monitoring, or to miss anything while waking from sleep).

Power Control

Often, the ELM327 is connected to a vehicle for only a short time, so power consumption is not of great concern. Occasionally, the ELM327 may be connected for longer times, however, possibly without the engine running. For those applications, it is often desirable to be able to put the circuit into a low power 'standby' state, and have it return to normal operation when needed. The power control features of the ELM327 were introduced for this.

There are four ways in which the ELM327 can be placed into the low power standby mode (these are shown pictorially in Figure 6). None of them will work without having the master enable (ie bit 7 of PP 0E) set to '1', which it is by default.

The first method is with an AT command. You may simply send:

```
>AT LP
```

and the IC will go to the low power mode after a one second delay (which allows the controlling circuit a little time to perform some housekeeping tasks).

When in Low Power mode, the ELM327 sets all outputs to their recessive/off state, sets pin 3 (J1850 voltage control) to a low level, and it reverses the state of the pin 16 control output. The ELM327 will then reduce its own power level, and begin monitoring for inputs that would cause a shift back to full power.

The next method allows automatic switching to the low power mode when there has been no RS232 input for a period of time - ie the controlling computer has stopped for some reason. To enable this method, both b7 and b5 of PP 0E must be set to '1'. The time delay (either 5 or 20 minutes) is set by b4, and the printing of a warning is enabled with b3. The warning is handy in some cases – it is the activity alert message ('ACT ALERT') and is sent 1 minute before the timer is about to time out. When the timer does time out, you will see a low power alert warning ('LP ALERT'), and then 2 seconds later, all of the outputs will change as described above for the AT LP command.

In a similar way, the Activity Monitor may be used

Power Control (continued)

to cause the ELM327 to shift to Low Power operation, if there is no OBD activity for some time. If you follow the logic path, you will see that b7 and b5 of PP 0F must both be '1', as well as b7 of PP 0E in order to allow this switch to occur. The default time that it allows before switching is set by b4, unless you have set a value with the AT AMT command. Setting the Activity Monitor Timeout to 00 blocks all the Activity Monitor outputs.

The final way to enter the low power mode is by a low level appearing at the ignition monitor input (pin 15), with both b2 and b7 of PP 0E set to '1'.

The ignition monitor logic inserts a short internal delay ('debounce') to be sure that the low level seen is a legitimate 'key off' and not just some noise. After it is sure, the ELM327 will then send a low power alert message ('LP ALERT'), and 2 seconds later it will switch to Low Power operation, just the same as was

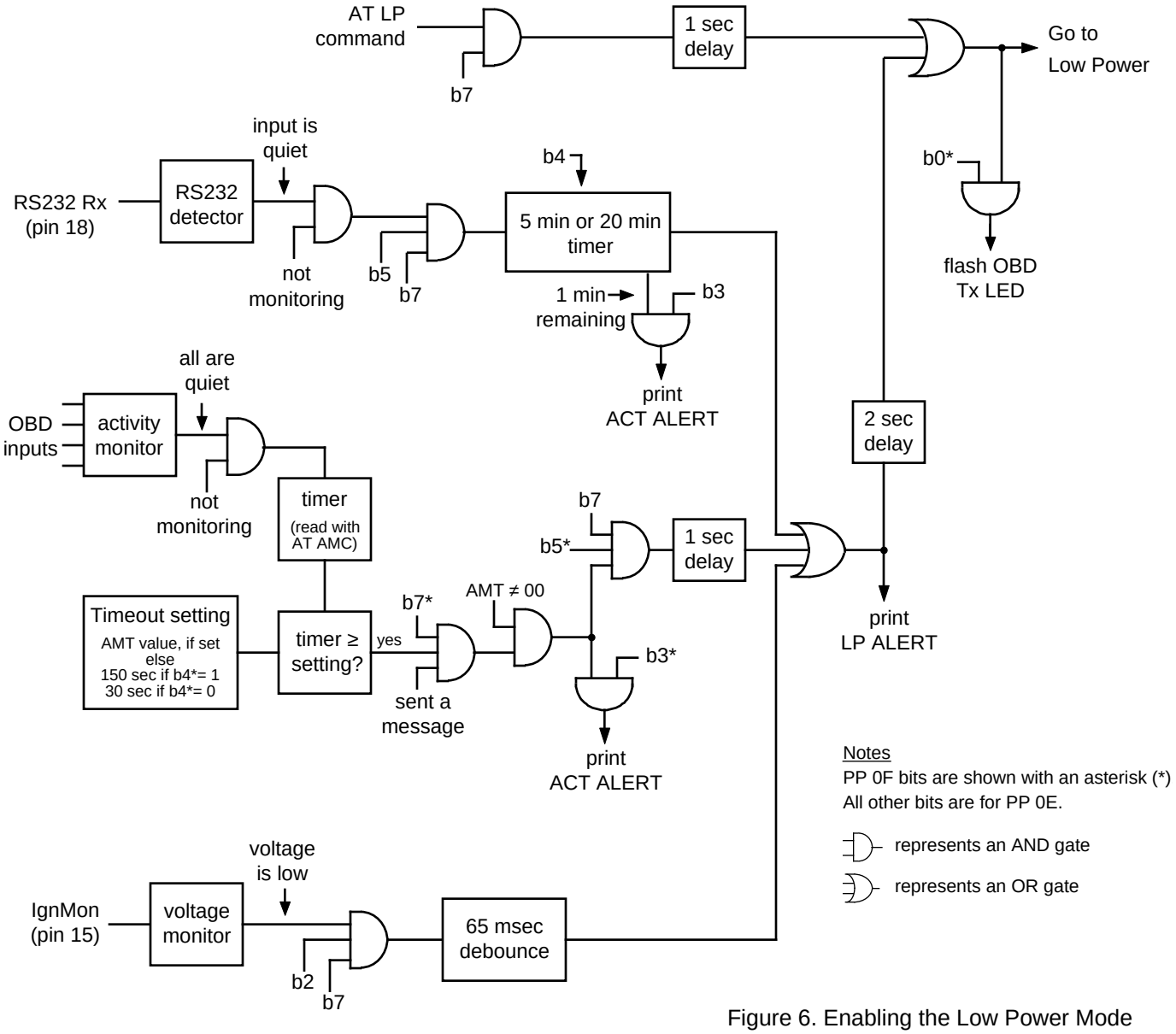
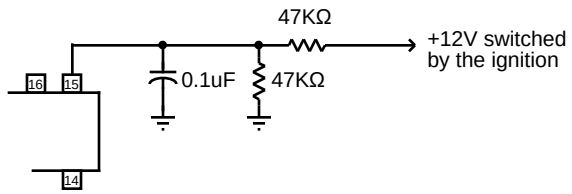


Figure 6. Enabling the Low Power Mode

Power Control (continued)

described for the other methods.

When connecting to pin 15, care should be taken to not allow excessive current (ie >0.5 mA) to pass through the internal protection diodes. Typically all that is required is a series resistor, but adding a capacitor helps to filter out ignition noise (note that the Schmitt input on pin 15 allows the use of large value capacitors). A second resistor ensures a discharge path for the capacitor, while raising the threshold voltage. A circuit like this works well:



Note that the AT IGN command can always be used to read the level at pin 15, regardless of the setting of the PP 0E enable bits. This may be used to advantage if you wish to manually shut down the IC, using your own timing and criteria. Recall that the alternate function for pin 15 is the RTS input which will interrupt any OBD processing that is in progress. If the ELM327 reports an interrupt with the 'STOPPED' message, you can then check the level at pin 15 with the AT IGN command, and make your own decisions as to what should be done. For that matter, you don't even need to reduce the power based on the input - you might possibly do something entirely different.

Having put the ELM327 into Low Power mode, you will need a method to wake it up. This is done by 'interrupting' the IC in ways that are very similar to that used to put it into Low Power mode.

Figure 7 shows the three ways to 'wake' the ELM327 from low power mode (other than toggling the power, or pulsing pin 1). Any of these can wake the IC - they do not have to be the cause of its going to low power.

The first way is with a low level pulse at the RS232 Rx input (pin 18). The RS232 circuitry is not as sensitive as normal when in the low power mode, so to be sure that your input is seen, the ELM327 requires that the pulse width be at least 128 usec wide. This is

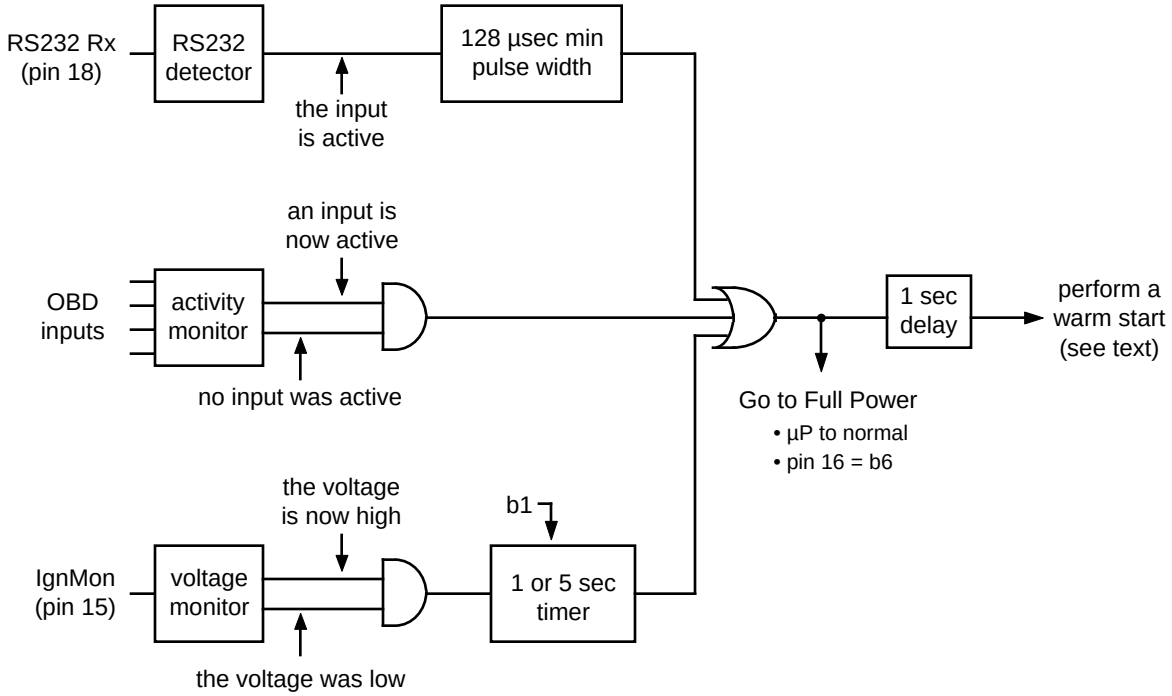


Figure 7. Returning to Normal Operation

Power Control (continued)

easily accomplished by sending a space or @ character, if the baud rate is 57.6 kbps or less. If using higher baud rates, you may wish to consider temporarily shifting to a lower baud rate and sending a character, or possibly see if your software can generate a 'break' signal for you. The break signal is not always the same length, but is typically several msec long (ie much longer than 128 µsec), and can often be sent by USB through USB to RS232 conversion devices. You will have to experiment if using other methods (Bluetooth or WiFi, for example).

The second method to have the ELM327 circuit go back to full power is by having activity appear on the OBD input pins. At the first sign of an active level, the circuit will start its wakeup, and within seconds, will be at full capability. Note that the logic is configured so that it must see all quiet on the OBD inputs before it will allow an automatic wakeup. This does not normally present a problem, and is implemented so that you can manually put the circuit to low power (AT LP), even if there is activity on one of the OBD inputs. Otherwise, the circuit would wake up immediately after the AT LP.

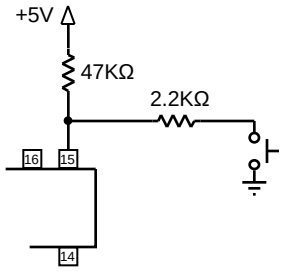
When connected to protocols 3, 4 or 5 (ISO 9141-2 or ISO 14230-4), there may be no OBD activity when the vehicle is first turned on. In this case, you could not rely on the activity monitor to wake the circuit and you would have to look for other means.

The final method that is provided to wake up the circuit is by way of the ignition monitor input and logic. A low then high level at the IgnMon input will cause the ELM327 to return to full power operation. Note that PP 0E bit 2 does not have to be set for the IgnMon to wake the circuit - the ELM327 always monitors this pin, and will wake the circuit after the delay that is set by PP 0E bit 1.

We are often asked if a switched 'ignition' voltage is available at the vehicle's diagnostic connector for this purpose. It is not, so you must connect a separate wire yourself. Often there are convenient places that you can use for this - possibly the radio power, or the connection to a convenience outlet.

Note that we present the pin 15 input as if it is an ignition monitoring circuit, but in fact, it will monitor for any change in voltage level (as long as the change is stable for the debounce period). You might consider connecting to other inputs (such as the 5V from a USB cable, for example) - just be sure to protect the ELM327 input circuitry from transients. This usually just requires a series resistor of 1K to 10K in value.

One alternative is to provide a pullup resistor to allow waking of the ELM327 with a momentary pushbutton switch:



If the circuit is in low power mode, a momentary push of the button will wake it up. This is just one of the many ways in which you can use this input.

One final note has to do with the changes to the startup process, beginning with firmware version 2.0. Initially (with v1.4 and v1.4b), the ELM327 simply used a warm start command (AT WS) to exit from the low power mode. The positive feedback that we received from our ELM329 changes has led us to also change the wakeup process for the ELM327.

Now, on switching back to full power operation, the ELM327 performs a warm start, but retains the following settings:

E0/1	H0/1	L0/1	M0/1
R0/1	D0/1	S0/1	AT0/1/2
CAF0/1	CFC0/1	CSM0/1	CEA
CER*	CTM1/5	JTM1/5	AL/NL
IIA		*only if CEA was active	

In addition, the ISO/KWP baud rate is retained and the current protocol is not changed (but it is closed, so will require initialization).

This has discussed how to have the ELM327 go to low power operation, and wake from it. While in the low power mode, the question of 'just how low is the power consumption' arises. We discuss that in the 'Modifications for Low Power Standby Operation' section, which is on page 85.

Programmable Parameters

The ELM327 contains several memory locations that retain their data even after power is turned off. Every time the IC is reset, 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. Because they define the operation of the ELM327, we refer to these memory locations as the Programmable Parameters.

All of the Programmable Parameters are easily altered at any time using a few simple commands. These 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 otherwise cause changes.

As an example of how to change a Programmable Parameter, consider PP 01 (shown on page 70) which sets the default state for the AT H command. If you are constantly powering your ELM327 and then using AT H1 to turn the headers on, you may want to change the default setting, so that the headers 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 yet enable 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 current AT H1/H0 setting. From the 'Type' column in the table on page 70, you can see that PP 01 is a type 'D' parameter, so 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. If you send the command AT D, then you should find that the headers are now displayed by default.

As there are many Programmable Parameters that you can alter, it might occasionally be difficult to know what changes you have made to them. To help with that, the ELM327 provides a Programmable Parameter Summary (PPS) command. This command simply prints a list of the entire range of PPs (whether currently supported or not), their current value, and whether they are on/enabled (N), or off/disabled (F). For a version 2.2 ELM327, with only the headers enabled (PP 01, as discussed above), the summary table would look like:

```
>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:9A F  0F:D5 F
10:0D F  11:00 F  12:FF F  13:55 F
14:50 F  15:0A F  16:FF F  17:6D F
18:31 F  19:4F F  1A:FF F  1B:FF F
1C:03 F  1D:0F F  1E:4A 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:3C 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), while the others are all off.

Another example shows how you might change the CAN filler byte. Some systems use 'AA' as the value to send for unused CAN data bytes, while the ELM327 uses '00' by default. To change the ELM327 in order to have it send AAs instead, 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 defaults are restored, whether it be by an AT D command, or by resetting the ELM327.

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 carriage return character has been redefined, or if the CAN response shows a data length code digit, 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, should you have to. 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

Programmable Parameters (continued)

rate, or the Carriage Return character, for example, and forget what you have set them to. The Carriage Return value that is set by PP 0D is the only character that is recognized by the ELM327 as ending a command, so 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 special 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 default settings. 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 an 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 flashing LED means that it worked and the PPs are all off.

This feature should only be used when you get into serious trouble, and it's your only choice (since putting a jumper into a live circuit might cause damage if you put it into the wrong place). Be careful if you need to use it.

Programmable Parameter Summary

The following pages provide a list of the currently available Programmable Parameters. The character shown in the 'Type' column indicates when changes 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 a 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)

PP	Description	Values	Default	Type
00	Perform an AT MA command 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 is high (logic 1)
baud rate (in kbps) = 4000 ÷ (PP 0C value)
For example, 500 kbps requires a setting of 08 (since 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)</th></tr>
<tr><td>19.2</td><td>D0</td></tr>
<tr><td>38.4</td><td>68</td></tr>
<tr><td>57.6</td><td>45</td></tr>
<tr><td>115.2</td><td>23</td></tr>
<tr><td>230.4</td><td>11</td></tr>
<tr><td>500</td><td>08</td></tr>
</table>
Notes:
1. The PP 0C value must be provided as hex digits only. The decimal values (listed above in brackets) are only shown for your convenience.
2. The ELM327 can only process continuous byte receives at rates of about 600 kbps maximum. If you need to connect at a higher rate, add a delay between the bytes to maintain an average rate of 600 kbps or less.
3. A value of 00 provides a baud rate of 9600 bps.</td><td>00 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>0E</td><td>Power Control options
Each bit controls an option, as follows:
b7: Master enable 0: off 1: on
if 0, pins 15 and 16 perform as described for v1.0 to v1.3a (must be 1 to allow any Low Power functions)
b6: Pin 16 full power level 0: low 1: high
normal output level, is inverted when in low power mode
b5: Auto LP control 0: disabled 1: enabled
allows low power mode if the RS232 activity stops
b4: Auto LP timeout 0: 5 mins 1: 20 mins
no RS232 activity timeout setting
b3: Auto LP warning 0: disabled 1: enabled
if enabled, says ‘ACT ALERT’ 1 minute before RS232 timeout
b2: Ignition control 0: disabled 1: enabled
allows low power mode if the IgnMon input goes low</td><td>00 to FF</td><td>9A
(10011010)</td><td>R</td></tr>
</table>

Programmable Parameter Summary (continued)

PP	Description	Values	Default	Type
0E	Power Control options (continued) b1: Ignition delay 0: 1 sec 1: 5 sec delay after IgnMon (pin 15) returns to a high level, before normal operation resumes b0: reserved for future - leave set at 0			
0F	Activity Monitor options Each bit controls an option, as follows: b7: monitor master control 0: disabled 1: enabled must be 1 to allow b3 to b6 b6: allow wake from Low Power 0: no 1: yes wakes on shift from no activity to activity b5: Auto LP control 0: disabled 1: enabled allows low power mode if the OBD activity stops b4: Auto LP timeout 0: 30 secs 1: 150 secs no OBD activity timeout setting b3: Auto LP warning 0: disabled 1: enabled if enabled, says 'ACT ALERT' on timeout b2: reserved for future - leave set at 1 b1: add exclamation mark 0: no 1: yes if 1, sends '!' before ACT ALERT and LP ALERT b0: LP LED 0: disabled 1: enabled if 1, the OBD Tx LED flashes when in Low Power mode (one 16 msec flash repeated every 4 seconds)	00 to FF	D5 (11010101)	D
10	J1850 voltage settling time setting (in msec) = (PP 10 value) x 4.096	00 to FF	0D (53 msec)	R
11	J1850 Break Signal monitor enable (reports BUS ERROR if break signal duration limits are exceeded)	00 = ON FF = OFF	00 (ON)	D
12	J1850 Volts (pin 3) output polarity normal = Low output for 5V, High output for 8V invert = High output for 5V, Low output for 8V	00 = invert FF = normal	FF (normal)	R
13	Time delay added between protocols 1 & 2 during a search setting (in msec) = 150 + (PP 13 value) x 4.096	00 to FF	55 (498 msec)	I
14	ISO/KWP final stop bit width (provides P4 interbyte time) setting (in µsec) = 98 + (PP 14 value) x 64	00 to FF	50 (5.2 msec)	D
15	ISO/KWP maximum inter-byte time (P1), and also used for the minimum inter-message time (P2). setting (in msec) = (PP 15 value) x 2.112	00 to FF	0A (21 msec)	D

Programmable Parameter Summary (continued)

PP	Description	Values	Default	Type
16	Default ISO/KWP baud rate (AT IB default setting) Note: 4800, 12500, and 15625 baud can not be set as defaults	00 = 96 FF = 10	FF (10.4K)	R
17	ISO/KWP wakeup message rate (AT SW default setting) setting (in msec) = (PP 17 value) x 20.48	00 to FF	92 (3.0 sec)	D
18	ISO/KWP delay before a fast init, if a slow init has taken place setting (in msec) = 1000 + (PP 18 value) x 20.48	00 to FF	31 (2.0 sec)	I
19	ISO/KWP delay before a slow init, if a fast init has taken place setting (in msec) = 1000 + (PP 19 value) x 20.48	00 to FF	4F (2.6 sec)	I
1A	Protocol 5 fast initiation active time (TiniL) setting (in msec) = (PP 1A value) x 2.5	00 to FF	0A (25 msec)	D
1B	Protocol 5 fast initiation passive time (TiniH) setting (in msec) = (PP 1B value) x 2.5	00 to FF	0A (25 msec)	D
1C	ISO/KWP outputs used for initiation (b7 to b2 are not used) b1:L line (pin 22) 0: disabled 1: enabled b0: K line (pin 21) 0: disabled 1: enabled If disabled, an output will remain low during protocol initiations	00 to FF	03 (00000011)	D
1D	ISO/KWP P3 time (delay before sending requests) Ave time (in msec) = (PP 1D value - 0.5) x 4.096	00 to FF	0F (59 msec)	D
1E	ISO/KWP K line minimum quiet time before an init can begin (W5) setting (in msec) = (PP 1E value) x 4.096	00 to FF	4A (303 msec)	D
1F	KWP byte count includes the checksum byte?	FF = NO 00 = YES	FF (NO)	R
21	Default CAN Silent Monitoring setting (for AT CSM)	FF = ON 00 = OFF	FF (ON)	R
24	CAN auto formatting (AT CAF default setting)	00 = ON FF = OFF	00 (ON)	D
25	CAN auto flow control (AT CFC default setting)	00 = ON FF = OFF	00 (ON)	D
26	CAN filler byte (always used to provide the extra CAN message bytes when 8 are required, whether formatting is on or not)	00 to FF	00	D
28	CAN Filter settings (controls CAN sends while searching) The bits of this byte control options, as follows: b7: 500 kbps match 0: ignored 1: required b6: 250 kbps match 0: ignored 1: required b5 to b1: reserved for future - leave set to 1 b0: send if bus is quiet 0: not allowed 1: allowed	00 to FF	FF (11111111)	D
29	Printing of the CAN data length (DLC) when printing header bytes (AT D0/D1 default setting)	00 = ON FF = OFF	FF (OFF)	D

Programmable Parameter Summary (continued)

PP	Description	Values	Default	Type
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 Processing 7F xx 78's: b2: enabled (CAN & KWP) 0: no 1: yes b1: valid Modes (xx values) 0: all 1: only 00 to 0F b0: valid CAN protocols 0: all 1: only ISO15765	00 to FF	3C (00111100)	D
2B	Protocol A (SAE J1939) CAN baud rate divisor baud rate (in kbps) = 500 ÷ (PP 2B 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: baud rate multiplier 0: x1 1: x 8/7 (see note 3) b3: reserved for future - leave set at 0. b2, b1, and b0 determine the data formatting options: 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 updates – results will be unpredictable if you should select one of them.	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

Notes:
1. The ELM327 does not accept decimal digits for the Programmable Parameters - all values are hexadecimal.
2. For Programmable Parameters that describe options in terms of bits, b7 is the msb, and b0 is the lsb.
3. When b4 of PP 2C or PP 2E are set, the CAN baud rate will be increased by a factor of 8/7, but the baud rate displayed by the AT DP command will still show the base rate (as set by PP 2D or PP 2F). For example, if you set PP 2C b4 to 1, and then PP 2D to 06, the base frequency will be 83.3 kbps. The AT DP command will report 83 kbps, but the actual baud rate will be 83.3x8/7 = 95.2 kbps.

Maximum CAN Data Rates

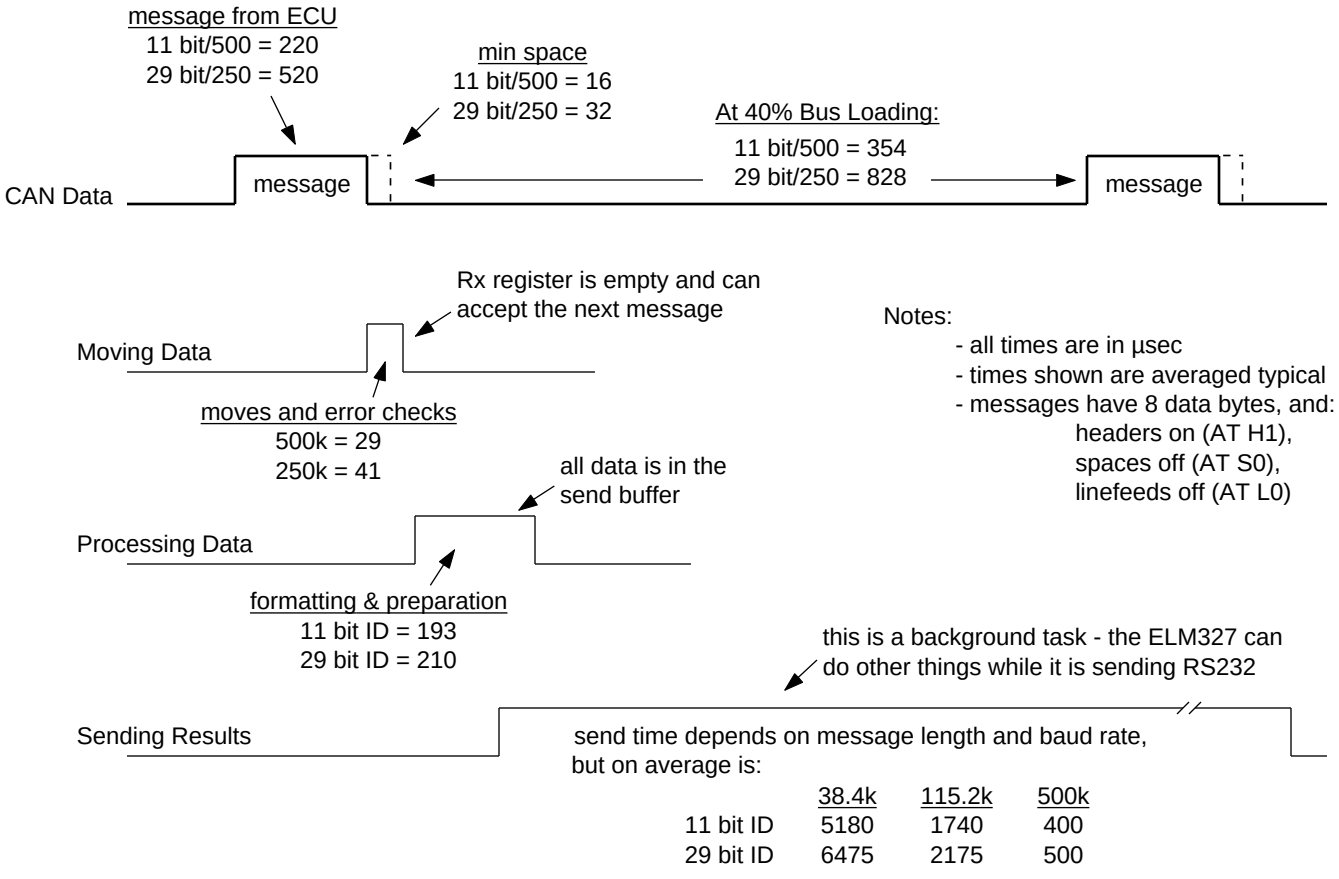
We are occasionally asked what the maximum data rate is that the ELM327 can handle. This is often after someone has tried to monitor all data using the default settings and has received a 'BUFFER FULL' error. It is difficult to say exactly what the maximum rate is, however, as several factors are involved.

The CAN 'engine' inside the ELM327 is actually configured with one receive register that accepts messages from the data bus, and another register that accepts messages from the first. As long as the firmware empties the second register before the first register needs it, there should not be any overflow problems with this component. The ELM327 actually moves the data quickly to temporary storage, so this is never a problem.

It would be nice if all the firmware had to do was to empty the second register, and wait for it to fill again, but that is not so. It must also check for errors, possibly queue a CAN response, format the received message, convert it to ascii, load it into the RS232 transmit buffer, and then prepare to receive the next message. These tasks can take a considerable time, depending on what formatting options you have chosen, and the baud rate that you select.

The diagram below shows these processes grouped into blocks. The times shown are typical for an ISO15765-4 message, and as you can see, vary with both the length of the CAN message and the CAN baud rate. All values shown are time measurements in µsec (microseconds).

When a message arrives, the ELM327 moves quickly to move the received bytes from the special CAN registers, so that they do not affect the next message that arrives. The data is then formatted (as ASCII bytes) and placed into the RS232 transmit buffer, for sending to the controlling processor. As long as messages do not arrive at a rate that is faster than the ELM327 can process them in, all messages will be



Maximum CAN Data Rates (continued)

processed. You can see from the figure that even for a 500 kbps message with an 11 bit ID, the ELM327 finishes with time to spare. Since ISO15765-4 specifies that messages must be 8 data bytes in length (filler bytes are added as needed) these times do represent the typical situation, with a 40% bus load. Actually, from these numbers the ELM327 should be able to handle 100% bus loading (which is not a practical situation).

Once the ELM327 has placed all of the properly formatted bytes into the RS232 transmit buffer, it is up to the controlling computer to fetch them in a timely fashion. If the bytes are removed too slowly, the buffer will continue to fill as new OBD messages arrive, and the buffer will eventually become full. It does not matter how big the buffer is, if the rate of removing bytes from the buffer is slower than the rate of putting them into the buffer, it will eventually fill up. When it is full, you will see a 'BUFFER FULL' message, and you will have to start over.

The ELM327 transmit buffer is 512 bytes in size. Considering that some bytes will be sent while new messages are being queued, this means that you can typically store:

	38.4k	115.2k	500k
11 bit/500k	28	38	–
29 bit/250k	26	56	–

messages in the buffer, if the bus loading is at 40%, as shown. This storage is more than enough for almost all OBD requests – the only time that you might get into trouble is if you are monitoring all messages on the bus (AT MA) with no filters set. In that case, you would need to be sure that you are removing bytes as fast as they are being generated.

The rate at which OBD messages occur depends on the 'bus loading'. This is a utilization factor that is very similar to the duty cycle for a square wave signal. Ideally, bus loading should be less than about 30%, but as vehicles become more complex, this is very difficult to do. Some vehicles are reportedly seeing 70% bus loads.

The above chart does not show any numbers in the 500 k column, as the buffer should never fill up when bus loading is 40%. As you increase loading, you will eventually reach the limit, but even with very busy data busses, we typically see about 150 messages before a BUFFER FULL is reported. If we turn off spaces and Linefeeds (AT S0, AT L0), we usually do not see any BUFFER FULL errors, even with very busy busses.

When people ask us then, 'What data rate can the ELM327 support?' the answer is not easy to provide, as it depends on many factors. It depends on the CAN message content, the data rate, and whether you have selected filters to eliminate irrelevant messages. The rate also depends on the RS232 baud rate that you have chosen, as it may be the limiting factor if there are a great number of messages being retrieved. If you are only fetching 10 or 20 messages, however, the internal buffer takes care of them for you, and the baud rate that you choose does not matter.

If you are simply fetching PID responses from the network, there really is no limitation within the ELM327, and it does not matter what the 'maximum CAN data rate' is. If you are trying to 'push the envelope', monitoring everything that travels on a very busy CAN data bus, then there will be limitations. What they are depend to a great deal on what choices you make. Hopefully this discussion will have helped to give you the necessary background information to do so.

Microprocessor Interfaces

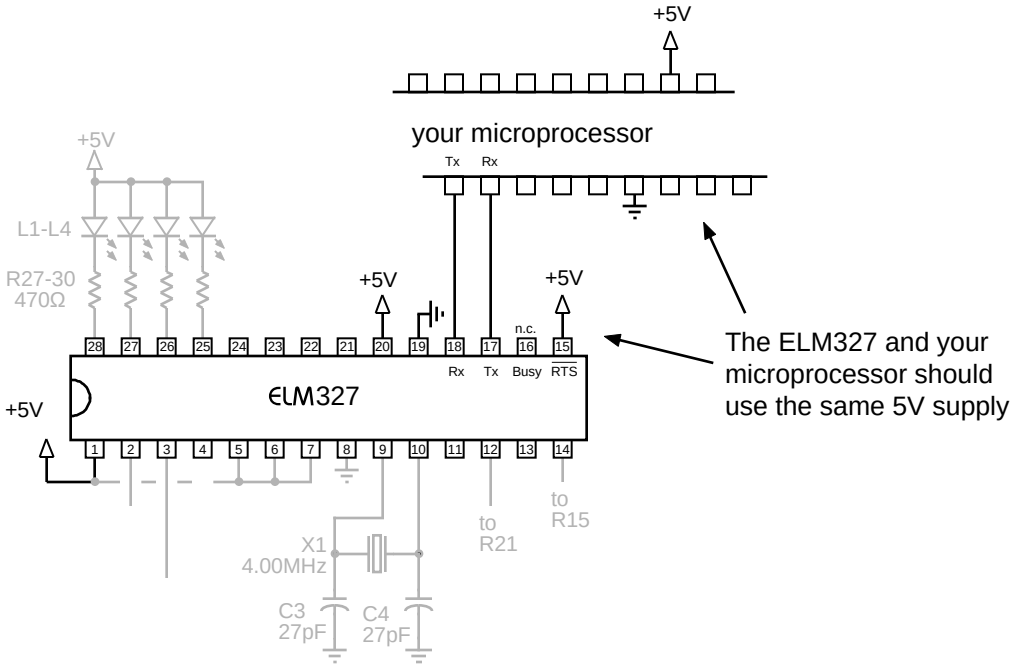
A very common question that we receive is ‘Can I connect the ELM327 directly to my own circuit, or must I use the interface shown?’ Certainly you may connect directly to our ICs, and you do not need to use an RS232 or USB interface. There are a few items to consider, however.

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 latch-up 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.

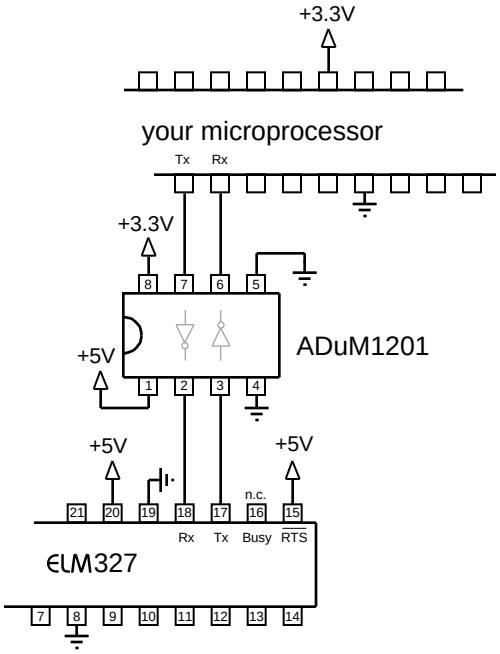
To use the handshaking feature, set one of your port pins to normally provide a high output, and connect it to the RTS input (pin 15). Use another port pin as an input to monitor the ELM327 Busy output (pin 16). When you want to send a command, simply check the Busy output first. If it is at a high logic 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. Note that



Microprocessor Interfaces (continued)

the default setting for PP 0E turns these hand-shaking signals off, so you will need to change that in order to use them.

We are often asked about connecting the ELM327 to 3.3V logic, which is quite popular. Many hope that they can just insert a resistor or two and have it work. Unfortunately, that is not the case, mostly because the ELM327 has Schmitt waveshaping on the RS232 Rx input (pin 18), and so may need as much as 4V for a high input (although it often works with less). To ensure that there's enough voltage, we recommend using a level translator circuit such as the TXB0102 from Texas Instruments (www.ti.com), the ST2129 from ST Microelectronics (www.st.com), or the Analog Devices (www.analog.com) ADuM1201 as shown here. We have been using the ADuM1201 with the Raspberry Pi lately, because it offers galvanic isolation (to 2500 Vrms) in addition to the level translation, so protects the Pi from occasional wiring problems. Note that for complete isolation you must use separate commons for the 5V and the 3.3V power supplies. The one disadvantage with the '1201 over the others is that it draws about 1 mA, which can be an issue if you are trying to use the low power mode with your ELM327.



Upgrading Versions

A popular question that we receive is "Can I upgrade my firmware with a file download?". The answer to this is no, the ELM327 can not be upgraded in this way - your integrated circuit must be replaced.

The next question which usually follows is "Can I simply replace an old ELM327 chip with a new one to upgrade the firmware?"

The answer to this last question is basically yes. We say basically because there was a change between versions 1.3a and 1.4 when we added the power control feature. This change modified how pins 15 and 16 were used (they took on dual roles), and that may affect your circuit.

If your circuit board is older and was designed for use with a v1.3a or older version of our ELM327 chip, then you have to look at what pins 15 and 16 were used for. Almost all of the early boards followed our example and left pin 16 open, and pin 15 tied to +5V. If your circuit did this, there is no problem - you may simply replace the old chip with a new one.

If your circuit board used pins 15 and 16 for handshaking with a microprocessor or other device, then you may need to take extra steps. Usually this only requires turning off the low power control by setting b7 of PP 0E to 0.

If you are replacing a v1.4b or a v2.0 integrated circuit with a newer one, then there are no concerns.

Please note that there are several clone products available that use integrated circuits that mimic the ELM327 at a software level. They do not necessarily mimic the hardware. For this reason, you should not replace a clone chip with a genuine ELM chip, unless you are absolutely certain that all 28 pins of the integrated circuits are identical.

Example Applications

The SAE J1962 standard dictates that all OBD compliant vehicles must use a standard connector, the shape and pinout of which is shown in Figure 8 below. The dimensions and pin connections for this 'Type A' connector are fully described in the SAE J1962 standard.

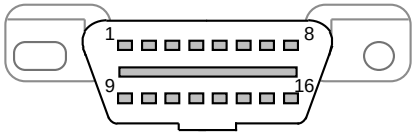


Figure 8. The J1962 Vehicle Connector

The circuit that you build with the ELM327 will be required to connect by way of a matching male J1962 connector. Fortunately these are available from several sources, easily found with a web search.

Note that before OBDII was adopted, several vehicles (most notably those made by General Motors) used a very similar connector (mostly) for their factory communications. These vehicles typically used what is known as the ALDL protocol, which the ELM327 does not support. Check that your vehicle is actually OBDII compliant before building your ELM327 circuit.

The circuit on page 81 (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 a capacitor for 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 (L5) for visual confirmation that power is present. We have used a 7805 regulator for this circuit as it is very common, and usually easy to obtain.

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 typically have a lot of critical information on them, and you can easily do more harm than good, so it is strongly recommended that you use a commercial transceiver chip as shown. We show a Microchip MCP2551 in this circuit, but most major manufacturers produce CAN transceiver ICs – look at the NXP PCA82C251, the Texas Instruments SN65LBC031, and the Linear Technology LT1796, to name only a few. Be sure to pay attention to the voltage limits as depending on the application, you may have to tolerate 24V, and not just 12V. Also, if you are considering using the Activity Monitor to wake the ELM327 from a low power sleep mode, be sure that the transceiver chip that you pick keeps the receiver functional when in standby mode (if it does not pass the signals on, the ELM327 can not see them).

The circuit shown directly below the CAN interface is used for the ISO 9141 and ISO 14230 signals. 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 do not connect anything to pin 22, and do not install Q6, R16 or R17.

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. Occasionally, we are asked about substitutes for these resistors – the standard specifies 510Ω but in a pinch you might be able to use 560Ω. A better solution would be to make 510Ω from 240Ω and 270Ω 1/4W resistors in series. We do not recommend using a lower value for the resistance as it stresses every device on the bus. Note that 1/2W resistors are specified in Figure 10 as a short at 13.8V causes about 0.4W dissipation.

Be careful if you are designing a circuit that might monitor other scan tools. Both the ELM327 and the other scan tool would present 510Ω resistors, so the vehicle would see 255Ω connected externally. This would very likely cause data errors, and might even damage some circuitry. To avoid this, you might wish to build your circuit in such a way that you can switch the ELM327's 510Ω resistors out, and replace them with a larger value. For example, you might put 10KΩ resistors in series with the 510Ω ones, and add removable jumpers or switches across the 10KΩ resistors. Most people do not need to do this – we only mention it because of the questions that we do receive.

The ISO data is both sent and received on the ISO-K line. Pin 12 of the ELM327 reads this data through the R20-R21 voltage divider. Because of the Schmitt trigger input on pin 12, these resistors will give typical threshold levels of 7.0V (rising) and 3.6V (falling), which provides a large amount of noise immunity while also protecting the IC. If you should connect test equipment to pin 12 (ie in parallel with R21), the thresholds will increase, so be conscious of

Example Applications (continued)

what you are doing while testing.

The final OBD interface shown is in the bottom left corner, and is used 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 8.0V and 5.5V, which works well for most vehicles. Note that the 317L is able to maintain regulation with a minimum of 1.5 mA of current, so we have used a 470Ω resistor between the output and adjust pins. The larger 317 regulator typically requires 3.5 mA so would need the resistors scaled down proportionally. Once suitable voltages have been generated, they are driven by the Q1-Q2 combination for the Bus+, and Q3 for the Bus-.

The J1850 VPW input uses a resistor divider, similar to that which 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. This is done by connecting Q4 across the input so that it operates as a difference amplifier. The Q4-D3 series combination sets a threshold voltage of about 1V (for improved noise immunity), while R11 limits the current flow, and R12 makes sure that Q4 is off when it should be. The circuit works well as shown, but the R14 passive pullup time constant can be easily affected by stray capacitance - be aware of this if connecting test equipment to pin 13.

Resistor R10 is the final J1850 component. We added this to help discharge the data bus more rapidly when it was found that some vehicles showed higher capacitance than others. The resistor may not be required for many vehicles - the choice is yours. If you should see reports of BUS ERRORs with a J1850 vehicle, it may be this capacitance causing problems (you will need to 'scope the signal to be sure).

Moving on, the R25-R26 voltage divider shown connected to pin 2 is used for the vehicle voltage monitoring circuitry. The two resistors simply divide the battery voltage to a safer level for the ELM327, and the capacitor (C2) helps to filter 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 your resistor values to maintain the input within the ELM327's V_{SS} to V_{DD} voltage range, and then perform an AT CV to calibrate the ELM327 to your new ratio. The maximum voltage that the ELM327 can show is 99.9V (it's a software limit, not hardware).

Four LEDs are shown on pins 25 to 28. These have been provided as a visual means of confirming circuit activity. They are not essential, but it is nice to see the visual feedback.

On the right side of the circuit, the ELM327's RS232 pins (17 and 18) are shown connected to an FTDI USB module. This module makes it very easy to connect the ELM327 circuit to your computer - all you need is the VCP Driver software, which is available for free from the FTDI web site (www.ftdichip.com). The module pinout matches a 9 pin D-sub connector, so you can simply solder it in where the RS232 circuitry used to go. Diode D5 and resistor R32 have been added to the interface to prevent the USB supply from backfeeding into the ELM327.

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 that you use. 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 and numbered all of the components and provided a summary parts list (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., then 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. For more help with building and testing the circuit, see our 'AN02 - ELM327 Circuit Construction' application note.

What if you only want to support one of the ELM327's protocols? Well, you may do that if you wish. Simply remove the OBD interfaces that you do not require, and connect the rest. Since you must never leave a CMOS input floating (open-circuited), you will need to add a jumper or two on the unused inputs. See the 'Unused Pins' section for advice on what to connect the pins to.

Some people still prefer to interface their circuits with standard RS232 circuitry. For these, we offer the

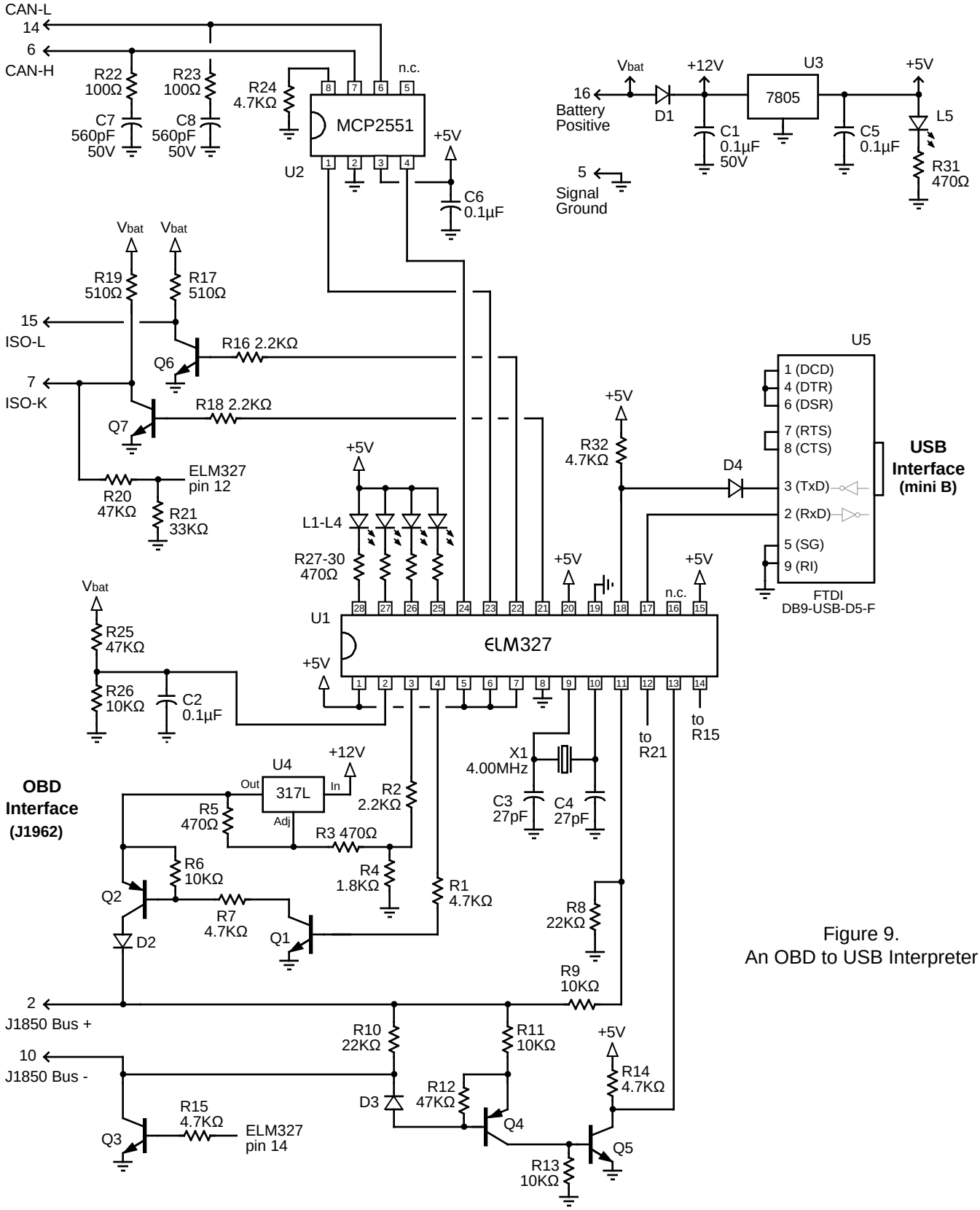


Figure 9.
An OBD to USB Interpreter

Example Applications (continued)

Semiconductors	Resistors (1/8W or greater, except as noted)
D1 = 1N4001	R22, R23 = 100 Ω
D2, D3, D4 = 1N4148	R3, R5, R27, R28, R29, R30, R31 = 470 Ω
L1, L2, L3, L4 = Yellow LED	R17, R19 = 510 Ω 1/2W
L5 = Green LED	R4 = 1.8 KΩ
Q1, Q3, Q5, Q6, Q7 = 2N3904 (NPN)	R2, R16, R18 = 2.2 KΩ
Q2, Q4 = 2N3906 (PNP)	R1, R7, R14, R15, R24, R32 = 4.7 KΩ
U1 = ELM327	R6, R9, R11, R13, R26 = 10 KΩ
U2 = MCP2551 or MCP2561	R10, R8 = 22 KΩ
U3 = 7805 regulator (5V 1A)	R21 = 33KΩ
U4 = 317L adjustable regulator (100 mA)	R12, R20, R25 = 47 KΩ
U5 = FTDI DB9-USB-D5-F usb module	
	Capacitors (16V or greater, except as noted)
Misc	C3, C4 = 27pF
X1 = 4.000MHz crystal	C7, C8 = 560pF 50V
DB9M connector for OBD cable?	C1 = 0.1uF 50V
IC Socket = 28 pin 0.3" wide (or 2 x 14pin)	C2, C5, C6 = 0.1uF

Figure 10. Parts List for Figure 9

sub-circuits of Figures 11 and 12.

Figure 11 shows a discrete RS232 interface, that may be connected directly to the ELM327. This circuit uses a resistor, diode, and capacitor between the two RS232 signal lines to 'steal' power from the host computer. In this way, the required RS232 negative voltage is obtained without adding a complicated power supply to the ELM327 circuit. The RS232 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. This circuit works well at baud rates of 57600 bps or less, but begins to show some errors at rates of 115200 bps and above.

The circuit of Figure 12 offers another RS232 solution that works well at higher baud rates. It uses a Maxim product (the MAX3222E) that is capable of operating at up to a 250 kbps rate (be sure to visit www.maximintegrated.com for more information).

The MAX3222E RS232 transceiver contains internal charge pump circuitry that generates the voltages required for RS232 communications, in addition to the analog interface circuitry needed. All you have to do is provide a few capacitors, and it does

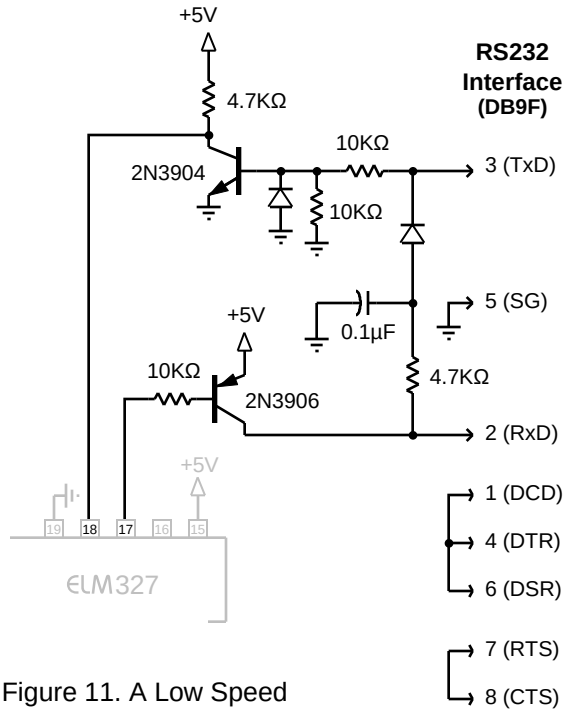


Figure 11. A Low Speed RS232 Interface (≤ 57.6 kbps)

Example Applications (continued)

the rest.

We do caution that the MAX3222E does seem to place some extra demands on the 5V power supply. It should work fairly well with the 7805 regulator that we show in Figure 9, but if you have chosen a lower current device like the 78L05 or LP2950, you may experience occasional 'LV RESET's. If you do, we recommend adding two capacitors to the circuit of Figure 9. We suggest that a 10 µF 50V capacitor be connected in parallel with C1, and also that a 33 µF 10V unit be installed across C5 (the 5V 'rail'). This should eliminate problems, but if it does not, you may also wish to consider a more powerful regulator (such as the 7805 that we show in Figure 9) if you do not already have one installed.

The USB interface of Figure 13 provides another way to connect the ELM327 to USB systems. It uses a Silicon Laboratories (www.silabs.com) CP2102 chip to convert between the ELM327's serial data and USB.

One of the advantages of going to a USB interface is the high serial baud rates that you may experience. In order to use these higher rates, you will have to program both the USB interface and the ELM327 interface for them.

The CP2102 interface baud rate is actually configured by driver software. When you set the baud rate in your terminal program, the software does what is necessary to configure the CP2102 for operation at that rate and you do not need to do anything more. The ELM327 initially only uses a 38,400 bps rate, however, and must be told to use anything different.

In order to change from the standard 38.4 k baud rate, you must first set your software to 38.4 kbps, and power up your ELM327 circuit. Make sure that it is working, as described in the 'Communicating with the ELM327' section, before you do anything else. When you are confident that all is well, you can then change the baud rate. Before doing so, we caution that you should check to be sure that your software actually supports the desired rate (as several can not handle more than about 250 kbps).

The CP2102 chip is able to support a 115.2 kbps rate natively, and almost all software should be able to support it as well, so we will use that rate to provide an example.

First, while connected to the ELM327 at 38.4 kbps, we need to change the default rate to 115.2 kbps. No need to worry that this will affect your communications, as it will not take effect until the ELM327 has been reset. To change the data rate, simply change the

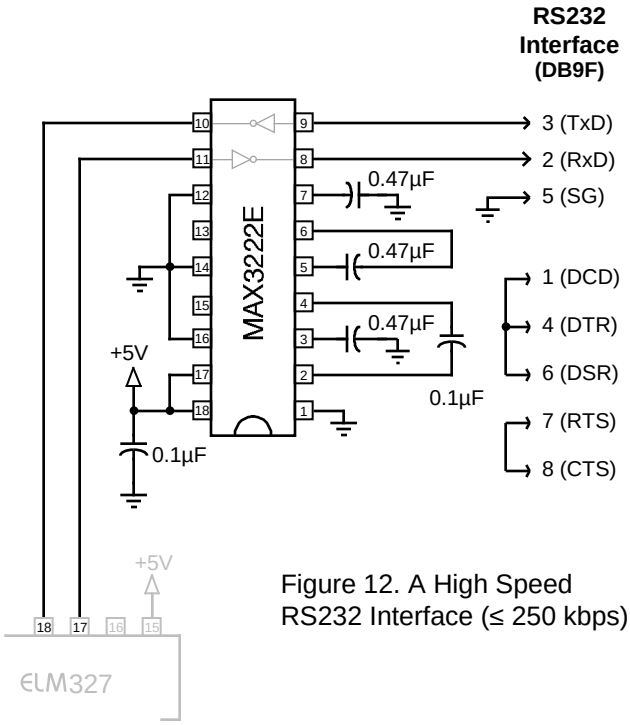


Figure 12. A High Speed RS232 Interface (≤ 250 kbps)

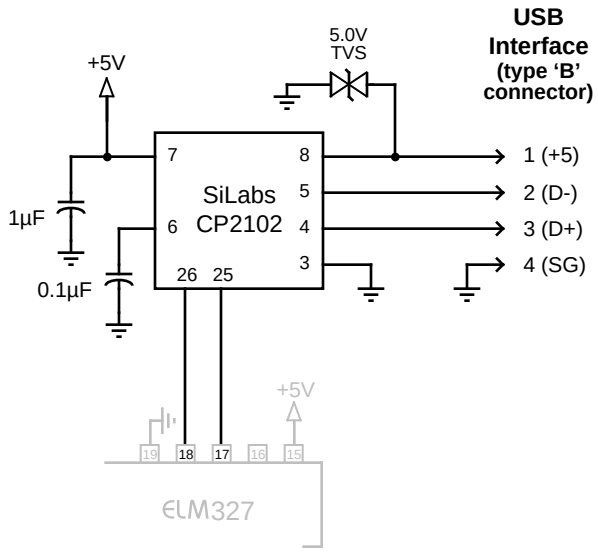


Figure 13. An Alternative USB Interface

Example Applications (continued)

value that is stored in Programmable Parameter 0C, then enable it:

```
>AT PP 0C SV 23
OK

> AT PP 0C ON
OK
```

That is all that is needed to semi-permanently change the ELM327. We say semi, because it stays in effect through power downs, and resets, but you can change it again if you wish. If you now reset the ELM327 (send AT Z or power down then up), the ELM327 will begin operating at the new rate (of 115200 bps, rather than to 38400 bps). Change your software setting to also be 115200 bps, and you should be communicating. If you go through the calculations, you will note that the ELM327 baud rate is actually off by about 0.8%, but modern UARTs can typically handle rate errors of a few % without any problems.

When working with 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 when choosing devices, as some exhibit a very large capacitance and will affect the transmission of the USB data.

Our final circuit (Figure 14) shows one way to interface the ELM327 to circuits that operate at a different voltage level. We show 3.3V as an example, but it can actually be anything from 2.7V to 5.5V.

The circuit uses the ADuM1201 iCoupler chip from Analog Devices (www.analog.com). In addition to acting as a level translator, this device also provides isolation (galvanic, to 2500 Vrms) between the two sides. This is often desired in order to keep the vehicle circuit completely separate from the computer circuit.

Typically, one might use a standard level shifter IC to interface to 3.3V – for example the TXB0102 by Texas Instruments (www.ti.com), or the ST2129 from ST Microelectronics (www.st.com), but the ADuM1201 offers several other advantages. The main difference is that it offers isolation, as mentioned, but it also can be used with 2.7V to 5.5V on either side, it provides a high output if the input side is unpowered, and it typically uses less current and is much faster than many opto-isolator solutions (the '1201 minimum data is 1 MHz). Of course there are a couple disadvantages too. It does use current (almost 1 mA), so may be an issue if trying to reduce sleep current to a minimum, and it does cost more than devices like the TXB0102.

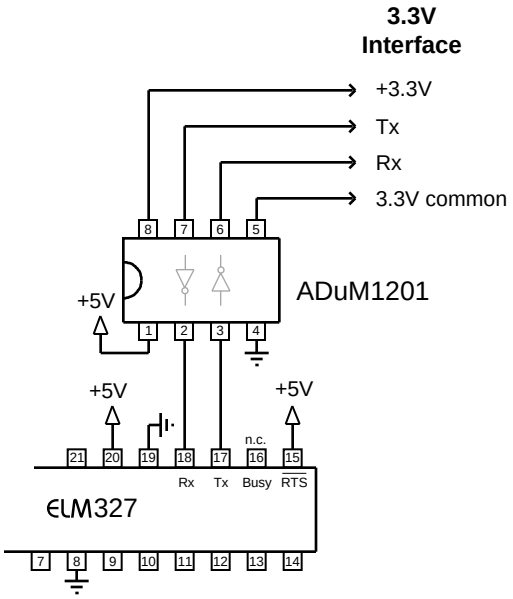


Figure 14. Connecting to a 3.3V System

Many wireless modules (WiFi or Bluetooth®) use serial interfaces just like what we have shown here for the RS232 connections. Connecting to them should not be very difficult if you follow the manufacturers directions (and perhaps consider using devices like the ADuM1201 or the TXB0102). If you are considering using a Bluetooth interface, you might read our 'AN-04 ELM327 and Bluetooth®' application note first.

This has provided a few examples of how the ELM327 integrated circuit might typically be used. Hopefully it has been enough to get you started on your way to many more. The following section shows how you might be able to optimize these circuits to reduce power consumption…

Modifications for Low Power Standby Operation

If you wish to semi-permanently install your ELM327 circuit in a vehicle, current consumption quickly becomes an issue. A typical ELM327 circuit draws about as much current as a dome lamp, so can not be left on for long without the vehicle's engine running.

When it is not needed, the ELM327 may be placed in a low power 'standby' mode in which it consumes very little current. Reducing the integrated circuit current itself is generally not enough, however, as you must also consider the current drawn by the other components as well.

The following discussion takes the circuit of Figure 9, and modifies it slightly in order to highlight our suggestions for reducing current consumption. The resulting circuit (Figure 15) is shown following. Note that portions of the circuit that are the same as Figure 9 are shown in grey, while the changes that we are making are shown in black.

Consider the circuit of Figure 9 in the Example Applications section. With 12V to 15V applied to the 'Battery Positive' input, the total current used by this circuit is typically:

base current = 29.8 mA

Without making any wiring changes, you can reduce this current by placing the ELM327 into the low power standby mode. This only requires sending the Low Power command (AT LP), after which the current would typically drop to:

current after AT LP = 17.6 mA

This reduction is due entirely to the change in the ELM327's operating current (it only needs a very small current to stay in standby mode). But where is the other current coming from? One obvious load is the LED that shows that the power is on. The other is the CAN transceiver IC, U2. By disconnecting the common connections from R24 and R31, and then returning both to pin 16 of the ELM327, we can switch the current that these two use. With this change (which is shown as modification #1 in Figure 15), the current after AT LP drops further:

current after mod #1 = 8.2 mA

There is a considerable current still flowing in the circuit at this point, but it should mainly be the voltage regulators that are using it. In the next step, we will change U3 (a 7805) to an LP2950ACZ-5.0G, and see how effective that is. While the LP2950 is a good choice for its lower quiescent current, it does suffer from stability problems if you do not provide capacitive loading as shown. Note that the 4.7uF capacitor is tantalum, while the 2.2µF one is aluminum. At this point it may also be useful to review our Application Note 'AN03 - ELM327 Low Voltage Resets', as you may want to use an even larger capacitor on the 5V side. After changing U3 for an LP2950, the current is typically:

current after mod #2 = 3.5 mA

If we continue to reduce the load currents beyond this point, we will quickly get to a point where any currents injected from external sources (ie. through the protection diodes at inputs such as pins 2 or 12) will become significant compared to the load currents. If these currents should exceed the load current, the Vdd voltage will rise and damage might result. To prevent the voltage from rising too high, we recommend that you add either a zener diode or a transient voltage suppressor (TVS), directly across the 5V supply, as shown in the top right corner of Figure 15. Suggestions for devices to consider are the 1N5232B zener diode or the SA5.0AG TVS.

Another integrated circuit that is not changing current during low power mode is the 317L (U4). In fact, a quick calculation shows that it is likely using about 2.6 mA when idle, which is very significant. If we replace this IC with another that uses less current, we will be close to getting the total circuit current to less than 1 mA.

Figure 15, modification #3 shows an LP2951ACM regulator in the circuit, as a replacement for the 317L. It uses much less current than the 317 during normal operation, and offers a shutdown control input as well to further reduce current when it is not required. Note that the LP2951 circuit needs the ELM327 to provide a high level at pin 3 for a 5V output, and a low for 8V, so an inversion is needed. To do this, set PP 12 to 00 with:

```
>AT PP 12 SV 00

>AT PP 12 ON
```

then reset the chip, and the voltages will always be correct for J1850 from that point on.

Note that the LP2951 regulator also requires a

Modifications for Low Power Standby Operation (continued)

4.7uF tantalum capacitor for stability (and it helps with transient capability too).

Leaving U4's pin 3 solidly connected to circuit common, we now find that the current is about:

current after mod #3 = 1.1 mA

and, if we tie pin 3 of the LP2951 to pin 16 of the ELM327, the LP current becomes:

current after mod #4 = 0.9 mA

We can suggest one more change at this point. The MCP2551 shown draws about 0.3 mA when in standby mode, and this can be improved upon. By replacing this chip with the newer MCP2561, the standby current will be reduced even further. Typically, you will see a change of at least 0.2 mA, giving:

current after mod #5 = 0.7 mA

The current has been reduced considerably through the circuit modifications, but why is there still current flowing? This is due to a number of things, some that you can change, others which you can not. The MCP2561 and the ELM327 are never completely shut off, but are instead in a low power mode, while the LP2950 regulator is operating normally (as the MCP2551 and the ELM327 need it). You can not do anything about that.

There are some currents that you may reduce, by choice in your design. The R25/R26 resistor pair uses current, but do you really need to monitor battery voltage? Similarly, the R20/R21 pair passes current, but does your application need ISO9141 or ISO14230 support? The OBD Tx LED flashes when in Low Power mode, but do you need it to? (You may turn it off with PP 0F b0, but it only uses about 25 µA on average). All of these little currents eventually add up to what we've measured here.

These few changes that we have shown have reduced the total current from about 30 mA to less than 1 mA (or power from 358 mW to 8 mW) - a considerable savings, which is enough for most applications. We leave any further improvements to you.

Modifications for Low Power Standby Operation (continued)

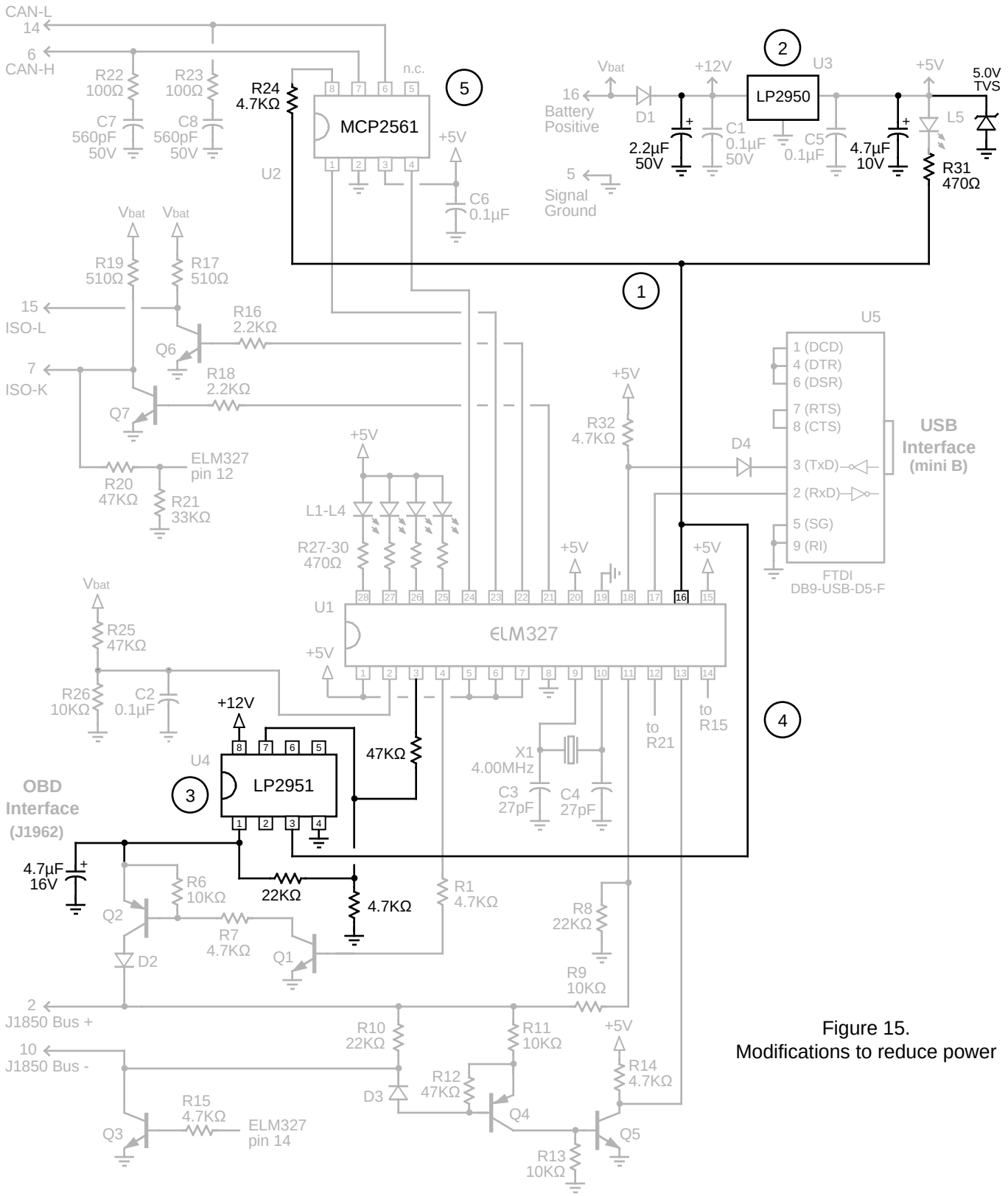


Figure 15.
Modifications to reduce power

Error Messages and Alerts

The following shows what the ELM327 will send to warn you of a condition or a problem. Some of these messages do not appear if using the automatic search for a protocol, or if the Programmable Parameter bits disable them.

?

This is the standard response for a misunderstood command received on the RS232 input. Usually it is due to a typing mistake, but it can also occur if you try to do something that is not appropriate (eg. trying to do an AT FI command if you are not set for protocol 5).

ACT ALERT or **!ACT ALERT**

This message occurs as a warning that there has been no RS232 or OBD activity for some time (see the Power Control section for details). If allowed, the IC will be initiating a switch to the Low Power (standby) mode of operation. If this was initiated by no RS232 activity, sending something within the next minute will stop the switch to low power. Note that the '!' before ACT ALERT is printed if PP 0F bit 1 is 1.

BUFFER FULL

The ELM327 provides a 512 byte internal RS232 transmit buffer so that OBD messages can be received quickly, stored, and sent to the computer at a more constant rate. Occasionally (particularly with CAN systems) the buffer will fill at a faster rate than it is being emptied by the PC. Eventually it may become full, and no more data can be stored (it is lost).

If you are receiving BUFFER FULL messages, and you are using a lower baud data rate, give serious consideration to changing your data rate to something higher. If you still receive BUFFER FULL messages after that, you might consider turning the headers and maybe the spaces off (with AT H0, and AT S0), or using the CAN filtering commands (AT CRA, or CM and CF) to reduce the amount of data being sent.

BUS BUSY

This occurs when the ELM327 tries to send a message, or to initialize the bus, and detects 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 almost always due to a wiring problem that is giving a continuously active input. If this is an initial trial with your new ELM327 circuit, then check all of the voltage levels at the offending OBD input, as this error is very likely due to a wiring problem (see our 'AN02 - ELM327 Circuit Construction' for some typical voltages).

BUS ERROR

A generic problem has occurred. This is most often from an invalid signal being detected on the bus (for example, a pulse that is longer than a valid Break signal), but usually is 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 baud 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

Error Messages and Alerts (continued)

ISO 15765-4 format. Try resending 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 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 previously made, as they will have been returned to their default values.

Beginning with v1.3a of this IC, an ERR94 will also block further 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.

LP ALERT or **!LP ALERT**

This appears as a warning that the ELM327 is about to switch to the Low Power (standby) mode of operation in 2 seconds time. This delay is provided to allow an external controller enough time to prepare for the change in state. No inputs or voltages on pins can stop this action once initiated. Note that the '!' before LP ALERT is printed if PP 0F bit 1 is 1.

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, and the IC stops all activity. In rare cases, a sudden large change in V_{DD} can also trigger a low voltage reset.

When the voltage returns to normal, the ELM327 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 v2.2).

An LV RESET will also block 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 seem to occur as a result of CAN wiring problems (the transceiver is capable of passing very large currents). 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 most often occurs if monitoring a CAN bus using an incorrect baud rate setting, but it may occur 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 message may not agree with the filter setting). Try a different protocol, or a different baud rate.

STOPPED

If any OBD operation is interrupted by a received RS232 character, or by a low level on the RTS pin, the

Error Messages and Alerts (continued)

ELM327 will print the word STOPPED. If you should see this response, then something that you have done has interrupted the ELM327. Most people see it because they have not waited for pin 15 to go high, or for the prompt character ('>') to be displayed before starting to send the next command.

Note that short duration pulses on pin 15 may cause the STOPPED message to be displayed, but may not be of sufficient duration to cause a switch to Low Power operation.

UNABLE TO CONNECT

If you see this message, it means that 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.

If you are sure that your vehicle uses an OBDII protocol, then check all of your connections, and the ignition, then try the command again.

Version History

We are often asked about the differences between the various ELM327 integrated circuits. The following summarizes some of these for you:

v1.0

Features:
- Supports SAEJ1850 PWM, SAEJ1850 VPW, ISO9141-2, ISO14230-4, and ISO15765-4 OBDII protocols.
- Communicates with a PC at 9.6 or 38.4 kbps

AT Commands:
@1, <CR>, AL, BD, BI, CAF0, CAF1, CF, CFC0, CFC1, CM, CP, CS, CV, D, DP, DPN, E0, E1, H0, H1, I, IB10, IB96, L0, L1, M0, M1, MA, MR, MT, NL, PC, R0, R1, RV, SH, SP, ST, SW, TP, WM, WS, Z

v1.0a

Features:
- Minor J1850 VPW timing adjustment for some 1999 and 2000 GM trucks.

v1.1

Features:
- Programmable Parameters
- User control over CAN flow control messages

AT Commands:
FC SD, FC SH, FC SM, PP FF OFF, PP FF ON, PP OFF, PP ON, PP SV, PPS

Programmable Parameters (new):
01, 02, 03, 09, 0A, 0D, 10, 11, 13, 16, 17, 18, 24, 25, 26, 29

v1.2

Features:
- RS232 baud rates are adjustable to 500 kbps
- Programmable Parameters can all be reset with a jumper
- Introduced Adaptive Timing
- Added SAE J1939 support (protocol A)
- Added user defined CAN protocols B and C
- KWP protocols allow four byte headers

AT Commands:
AR, AT0, AT1, AT2, BRD, BRT, DM1, IFR H, IFR S, IFR0, IFR1, IFR2, IIA, KW0, KW1, MP, SR, WM

Programmable Parameters:
00, 04, 06, 07, 0C, 2B, 2C, 2D, 2E, 2F

v1.2a

Features:
- Minor change to improve error detection

v1.3

Features:
- Added ability to state the number of responses desired
- New CAN CRA commands to help with setting the mask and filter
- Able to send CAN RTR messages
- New STOPPED message for user interrupts during searches
- Introduced LV RESET message for resets from low voltage
- New @2 and @3 commands for storing of a unique identifier

AT Commands:
@2, @3, CRA, D0, D1, JE, JS, KW, MP, RA, RTR, S0, S1, SP00, V0, V1

Programmable Parameters:
2A

v1.3a

Features:
- Added wiring checks to detect when the J1962 CAN pins are used for other functions

AT Commands:
FE

Programmable Parameters:
none

Version History (continued)

v1.4

Features:
- Added Low Power (sleep) mode
- Added extended addressing mode for CAN protocols
- Added 4800 baud ISO 9141 and ISO 14230 support
- ISO 9141 and ISO 14230 can be manually initiated
- Provided a single EEPROM byte for user data storage
- All interrupts now say STOPPED (not just when searching)

AT Commands:
CEA, CV 0000, FI, IB48, IGN, LP, PB, RD, SD, SI, SS, TA

Programmable Parameters:
0E, 12, 15, 19, 2C, 2E

v1.4a

Elm Electronics never made a v1.4a

v1.4b

Features:
- Able to provide active or passive CAN monitoring
- New CRA command to restore the mask and filter to their default values
- Several SAE J1939 improvements

AT Commands:
CRA, CSM0, CSM1, JHF0, JHF1, JTM1, JTM5, MP (with # messages)

Programmable Parameters:
21

v1.5

Elm Electronics never made a v1.5

v2.0

Features:
- Increased the RS232 Tx buffer to 512 bytes
- New Activity Monitor watches all OBD inputs
- Wake from Low Power now retains settings
- The CRA commands now accept X's for input
- New PP's provide extensive ISO/KWP control
- Brownout reset voltage reduced to 2.8V

AT Commands:
AMC, AMT, CRA (with X's), SH (4 byte)

Programmable Parameters:
0C, 0F, 14, 15, 19, 1A, 1B, 1C, 1D

v2.1

Features:
- Many optimizations for increased speed
- Detects response pending replies (7F xx 78) and adjusts timeouts for same
- CAN searches now measure frequency and block sends if there is a mismatch

AT Commands:
CTM1, CTM5

Programmable Parameters:
1E, 28

v2.2

Features:
- New ISO/KWP 12500 and 15625 baud rates
- New CER command allows defining the CEA receive address
- New IFR modes for controlling J1850 in frame responses while monitoring
- KWP data count can include the checksum
- CAN Status command now shows frequency

AT Commands:
IB12, IB15, CER, IFR4, IFR5, IFR6

Programmable Parameters:
1F, changed PP19 default value

Outline Diagrams

The diagrams at the right show the two package styles that the ELM327 is available in.

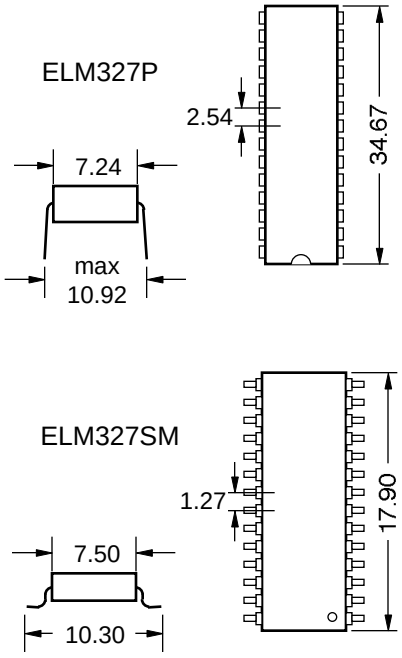
The first shows our ELM327P product in what is commonly called a '300 mil skinny DIP package'. It is used for through hole applications.

The ELM327SM package shown at right is also sometimes referred to as 300 mil, and is often called an SOIC package. We have chosen to simply refer to it as an SM (surface mount) package.

The drawings shown here provide the basic dimensions for these ICs only. Please refer to the following Microchip Technology Inc. documentation for more detailed information:

Package Drawings and Dimensions Specification,
(document name en012702.pdf - 7.5MB).
Go to www.microchip.com, select 'Design Support' then 'Documentation' then 'Packaging Specifications', or go directly to www.microchip.com/packaging

PIC18F2480/2580/4480/4580 Data Sheet,
(document name 39637d.pdf - 8.0MB).
Go to www.microchip.com, select 'Design Support' then 'Documentation' then 'Data Sheets, and search by Products for 18F2480.



Note: all dimensions shown are in mm.

Ordering Information

ELM327 integrated circuits are 28 pin devices, available in either a 300 mil wide ('skinny') plastic DIP format or in a 300 mil (7.50 mm body) SOIC surface mount type package. We do not offer an option for QFN packages.

To order, add the appropriate suffix to the part number:

300 mil 28 pin Plastic DIP.....ELM327P 300 mil 28 pin SOIC.....ELM327SM

Index

Index (continued)