ELM327

OBD to RS232 Interpreter

Description

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

The ELM327 builds on improved versions of our proven ELM320, ELM322, and ELM323 interfaces by adding four CAN protocols to them. The result is an IC that can automatically sense and convert the nine most common protocols in use today. There are a number of other improvements as well - a high speed RS232 option with data buffering, battery voltage monitoring, and the ability to remember the last used protocol, to name only a few.

The ELM327 requires few external components to make a fully functioning circuit. The following pages discuss the interface details, and show how to use the IC to 'talk' to your vehicle, before concluding with two typical schematics to get you started in the Example Applications section.

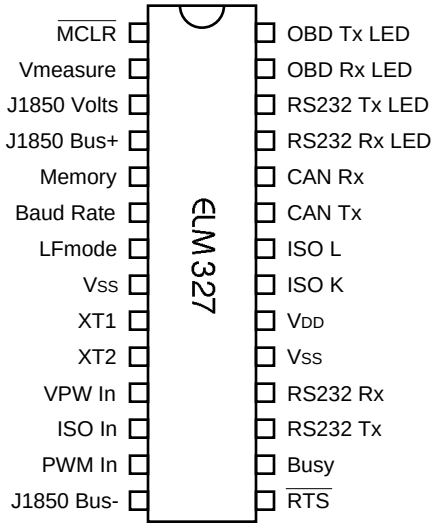
Applications

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

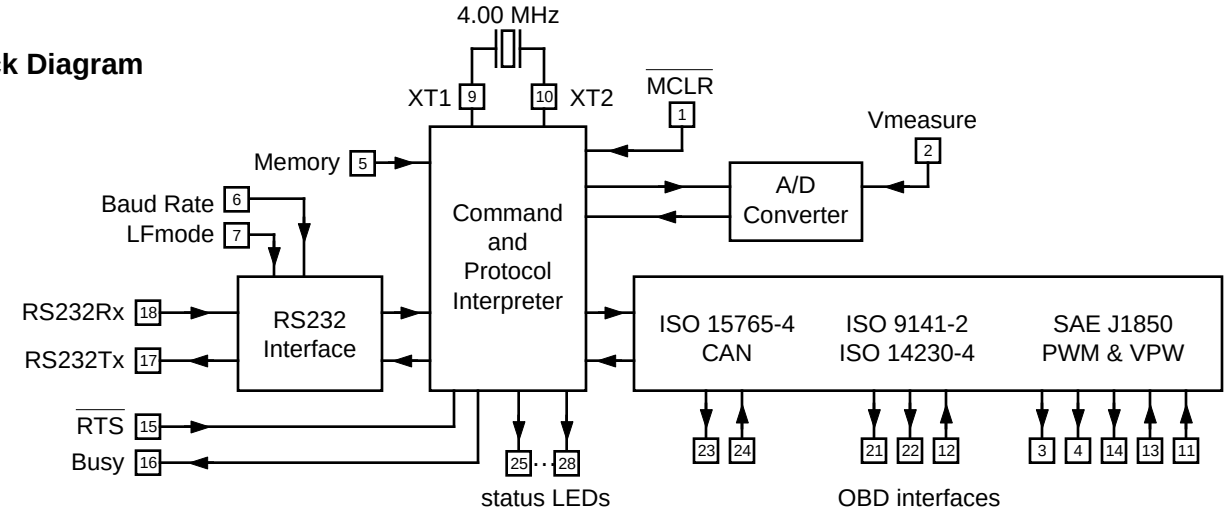
Features

- Supports 9 OBDII protocols
- Automatically searches for a protocol
- Fully configurable with AT commands
- High and Medium speed RS232
- Voltage input for battery monitoring
- Low power CMOS design

Connection Diagram
PDIP and SOIC
(top view)



Block Diagram



Pin Descriptions

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

A logic low applied to this input will reset the IC. 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.

J1850 Volts (pin 3)

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

J1850 Bus+ (pin 4)

This active high output is used 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 is enabled by default. If it is at a low level, then the default will be to have it disabled. Memory can always be controlled with the AT M1 and AT M0 commands at other times.

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. If at a low level, the baud rate will be 9600.

LFmode (pin 7)

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

V_{SS} (pins 8 and 19)

Circuit common must be connected to these pins.

XT1 (pin 9) and XT2 (pin 10)

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

VPW In (pin 11)

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

ISO In (pin 12)

This is the active low input for the ISO 9141 and ISO 14230 data signal. It is derived from the K Line, and should be at a high logic level when at rest (bus recessive). No special amplification is required, as this input has Schmitt trigger waveshaping.

PWM In (pin 13)

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

Pin Descriptions (continued)

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, the output can be left open-circuited.

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

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

Busy (pin 16)

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

RS232Tx (pin 17)

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

RS232Rx (pin 18)

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

V_{DD} (pin 20)

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

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

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

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

These are the two CAN interface signals that must be connected to a CAN transeiver IC for proper operation. If you are connecting to an existing CAN system, the integrity of that sysem might be jeopardized if a proper interface is not used. See the Example Applications section for more information.

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. Current capability is suitable for directly driving most LEDs through current limiting resistors, or interfacing to other logic for status reporting. If unused, these pins should be left open-circuited.

Ordering Information

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

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

300 mil 28 pin SOIC.....ELM327SM

Absolute Maximum Ratings

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

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

Voltage on V_{DD} with respect to V_{SS}..... 0 to +7.5V

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

Note:

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

Electrical Characteristics

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

Characteristic	Minimum	Typical	Maximum	Units	Conditions
Supply voltage, V_{DD}	4.5	5.0	5.5	V	
V_{DD} rate of rise	0.05			V/ms	see note 2
Average supply current, I_{DD}		9		mA	see note 3
Input threshold voltage	1.0		1.3	V	all except Schmitt inputs
Schmitt trigger input thresholds rising		3.0		V	see note 4
Schmitt trigger input thresholds falling		1.4		V	see note 4
Output low voltage		0.3		V	current (sink) = 10 mA
Output high voltage		4.6		V	current (source) = 10 mA
Brown-out reset voltage	4.07	4.2	4.59	V	
A/D conversion time		7		msec	see note 5

Notes:

1. This integrated circuit is produced with a Microchip Technology Inc.'s PIC18F248 or PIC18F2480 as the core embedded microcontroller. For further device specifications, and possibly clarification of those given, please refer to the appropriate Microchip documentation (available at http://www.microchip.com/).
2. This spec must be met in order to ensure that a correct power on reset occurs. It is quite easily achieved using most common types of supplies, but may be violated if one uses a slowly varying supply voltage, as may be obtained through direct connection to solar cells, or some charge pump circuits.
3. Device only. Does not include any load currents.
4. Pins 1, 11, 12, 13, 15 and 18 have internal Schmitt trigger waveshaping circuitry
5. The typical width of the Busy output pulse while the ELM327 interprets the command, measures the voltage, scales it and transmits the result of a mid-range measurement at 38400 baud.

Overview

The following describes how to use the ELM327 to obtain a great deal of information from your vehicle. To some, this information will be overwhelming, and for others it will be not nearly enough.

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

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

Communicating with the ELM327

The ELM327 relies on a standard RS232 type serial connection to communicate with the user. Ensure that you have chosen the proper data rate (either 9600 or 38400 baud), with 8 data bits, no parity bit, and 1 stop bit. All responses from the IC are terminated with a single carriage return character and, optionally, a linefeed character. Make sure your software is configured properly for the "line end" mode that you have chosen.

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

```
ELM327 v1.0

>
```

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 displayed above is the ELM327's prompt character. It indicates that the device is in its idle state, ready to receive characters on the RS232 port. Messages sent from the computer can either be intended for the ELM327's internal use, or for reformatting and passing on to the OBD bus.

The ELM327 can quickly determine where the received characters are to be directed by analyzing the entire string once the complete message has been received. Commands for the ELM327's internal use will always begin with the characters 'AT' (as is common with modems), while commands for the OBD bus are only allowed to contain the ASCII codes for hexadecimal digits (0 to 9 and A to F).

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

Incomplete or misunderstood messages can also occur if the controlling computer attempts to write to the ELM327 before it is ready to accept the next command. To avoid a data overrun, users should always wait for the prompt character ('>'), or a low on the Busy output before sending the next command.

Finally, there are a few convenience items to note. The ELM327 is not case-sensitive, so 'ATZ' is equivalent to 'atz', and to 'AtZ'. Also, it ignores space characters and all control characters (tab, linefeed, etc.) in the input, so they can be inserted anywhere to improve readability. Another feature is that sending only a single carriage return character will always repeat the last command (making it easier to request updates on dynamic data such as engine rpm).

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 issued.

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 internal configuration or 'AT' commands, and will be executed upon receipt of a terminating carriage return character. The ELM327 will usually reply with the characters 'OK' on the successful completion of a command, so the user knows that it has been executed.

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 may prove useful if you wish to interpret the values. Also, one should be aware that for the on/off types of commands, the second character is the number 1 or 0, the universal terms for on and off.

The following is a description of all of the AT commands that are recognized by the current version of the ELM327. Since there are many, a summary page is provided after this section.

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).

BD [perform an OBD Buffer Dump]

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

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

BI [Bypass the Initialization sequence]

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

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

These commands determine whether the ELM327 assists you with the formatting of the CAN data that is sent and received. With CAN Automatic Formatting enabled (CAF1), the IC will automatically generate formatting (PCI) bytes for you when sending, and will remove them when receiving. This means that you can continue to issue OBD requests (01 00, etc.) as usual, without regard to these extra bytes that the CAN diagnostics systems require. With formatting on, the trailing (unused) data bytes that are received in a frame will be removed as well, and only the relevant ones will be shown.

Turning the CAN Automatic Formatting off (CAF0), will cause the ELM327 to print all of the received data bytes. No bytes will be hidden from you, and none will be inserted for you. Similarly, when sending a data request with formatting off, you must provide all of the

AT Commands (continued)

required data bytes exactly as they are to be sent – the ELM327 will not perform any formatting for you other than to add some trailing 'padding' bytes to ensure that the required eight data bytes are sent. This allows operation in systems that do not use PCI bytes as ISO 15765-4 does.

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

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

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

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

CF hhh [set the CAN ID Filter to hhh]

The CAN Filter works in conjunction with the CAN Mask to determine what information is to be accepted by the receiver. As each message is received, the incoming CAN ID bits are compared to the CAN Filter bits (when the mask bit is a ‘1’). If all of the relevant bits match, the message will be accepted, and processed by the ELM327, otherwise it will be discarded. This three nibble version of the CAN Filter command makes it a little easier to set filters with 11 bit ID CAN systems. Only the rightmost 11 bits of the provided nibbles are used, and the most significant bit is ignored. The data is actually stored as four bytes internally, however, with this command adding leading zeros for the other bytes. See the CM command(s) for more details.

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

This command allows all four bytes (actually 29 bits) of the CAN Filter to be set at once. The 3 most significant bits will always be ignored, and can be given any value. Note that this command can be used to enter 11 bit ID filters as well, since they are stored in the same locations internally (entering AT CF 00 00 0h hh is exactly the same as entering the shorter AT CF hhh command).

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

The ISO 15765-4 protocol expects a “Flow Control” message to always be sent in response to a “First Frame” message. The ELM327 automatically sends these, and it is usually of little concern to the user. If experimenting with a non-OBD system, it may be desirable to turn this automatic response off. The AT CFC0 command has been provided for that purpose. The default setting is CFC1 - Flow Controls on.

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

CM hhh [set the CAN ID Mask to hhh]

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

Note that a common storage location is used internally for the 29 bit and 11 bit masks, so an 11 bit mask could conceivably be assigned with the next command (CM hh hh hh hh), should you wish to do the

AT Commands (continued)

extra typing. The values are right justified, so you would need to provide five leading zeros followed by the three mask bytes.

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

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

CP hh [set CAN Priority bits to hh]

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

CS [show the CAN Status]

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 them, the device can even go off-line in order not to affect other data on the bus, should there be a hardware or software fault. The AT CS command lets you see both the Tx and the Rx error counts. If the transmitter should be off (count >FF), you will see 'OFF' rather than a specific count.

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

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

To use this calibration feature, simply use a meter with sufficient accuracy to read the actual input voltage. If, for example, the ELM327 consistently says the voltage is 12.2V when you measure 11.99 volts, simply issue AT CV 1199, and the device will recalibrate itself for the provided voltage (it should then read 12.0V due to roundoff). If you use a test voltage that is less than 10 volts, don't forget to add a leading zero (that is, 9.02 volts should be entered as AT CV 0902).

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.

DP [Describe the current Protocol]

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

DPN [Describe the Protocol by Number]

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

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

These commands control whether or not characters received on the RS232 port are retransmitted (or echoed) back to the host computer. To reduce traffic on the RS232 bus, users may wish to turn echoing off by issuing ATE0. The default is E1 (echo on).

AT Commands (continued)

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

These commands control whether or not the additional (header) bytes of information are shown in the responses from the vehicle. These are normally not shown by the ELM327, but can be by issuing the AT H1 command.

Turning the headers on actually shows more than just the header bytes - you will see the complete message as transmitted, including the check-digits, and PCI bytes. The only exception is that the current version does not display the CAN data length code (DLC), the CRC, nor the special J1850 IFR bytes (which some protocols use to acknowledge receipt of a message).

I [Identify yourself]

Issuing this command causes the chip to identify itself, by printing the startup product ID string (currently "ELM327 v1.0"). Software can use this to determine exactly which integrated circuit it is talking to, without having to reset the IC.

IB 10 [set the ISO Baud rate to 10400]

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

IB 96 [set the ISO Baud rate to 9600]

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

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

This option controls the sending of linefeed characters after each carriage return character. If the ATL1 is issued, linefeeds will be generated after every carriage return character, and for ATL0, it will be off. Users will generally wish to have this option on if using a terminal program, but off if using a custom computer interface (as the extra characters transmitted will only serve to slow the communications down). The default setting is determined by the voltage at pin 7 during power on (or reset). If the level is high, then linefeeds on will be the default; otherwise it will be linefeeds off.

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

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

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

If the ELM327 is to be used in an environment where the protocol is constantly changing, it would likely be best to turn the memory function off, and issue an AT SP 0 command once. The SP 0 command tells the ELM327 to always 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 good sense.

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

MA [Monitor All messages]

Using this command places the ELM327 into a bus monitoring mode, in which it displays all messages as it sees them on the OBD bus. This continues indefinitely until stopped by activity on the RS232 input, or the RTS pin. To stop the monitoring, one can send a single character then wait for the ELM327 to respond with a prompt character ('>'). Alternatively, the RTS input can be brought to a low level to interrupt the

AT Commands (continued)

device as well. Waiting for the prompt is necessary as the response time is unpredictable, varying depending on what the IC was doing when interrupted. If for instance it is in the middle of printing a line, it will first complete the line then return to the command state, issuing the prompt character. If it were simply waiting for input, it would return immediately. Note that the character which stops the monitoring will always be discarded, and will not affect subsequent commands.

MR hh [Monitor for Receiver hh]

This command also places the IC in a bus monitoring mode, displaying only 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.

MT hh [Monitor for Transmitter hh]

Another monitoring command, which displays only messages sent by transmitter address 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 systems. As with the MA and MR monitoring modes, any RS232 activity (single character) aborts the monitoring.

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 default is NL on.

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.

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

These commands control the ELM327's automatic display of responses. 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. This is useful if sending commands blindly when using the IC for a non-OBD network application, or simulating an ECU in a basic learning environment. It is not recommended that this option normally be used, however, as the vehicle may have difficulty if it is expecting an acknowledgement byte and never receives one. The default is R1, or responses on.

RV [Read the input Voltage]

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

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

This command allows the user to manually control the values that are sent as the three header bytes in a message. These bytes are normally assigned values for you (and are not required to be adjusted), but there may be occasions when it is desirable to change them (particularly if experimenting with physical addressing). The value of hex digits xx will be used for the first or priority/type byte, yy will be used for the second or receiver/target byte, and zz will be used for the third or transmitter/source byte. These remain in effect until set again, or until restored to their default values with the D, WS, or Z commands.

This command is used to assign all header bytes, whether they are for a J1850, ISO 9141, ISO 14230, or a CAN system. The CAN systems will use these three bytes to fill bits 0 to 23 of the ID word (for a 29 bit ID), or will use only the rightmost 11 bits for an 11 bit CAN ID. The additional 5 bits needed for a 29 bit system are provided through the AT CP command (since they rarely change).

AT Commands (continued)

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

Entering an 11 bit ID word (header) normally requires that extra leading zeros be added (eg. AT SH 00 07 DF), but this command simplifies doing so. The SH xyz AT command accepts a three digit argument, takes only the right-most 11 bits from that, adds leading zeros, and stores the result in the header storage locations for you. As an example, AT SH 7DF is a valid command, and is quite useful for working with 11 bit CAN systems. It actually results in the header bytes being internally stored as 00 07 DF.

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.

Currently, the valid protocols are:

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

The Automatic selection (protocol 0) is a convenient way of telling the ELM327 to automatically try all protocols, when looking for a valid one. It will first try protocol 1, then will sequence through each of the others, until one is 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.

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

SP Ah [Set Protocol to Auto, h]

This variation of the SP command allows you to set 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 AT SP A3. The default protocol will then be 3, but with the ability to automatically search for other protocols. Don't forget to disable the memory function if doing this, or your neighbour's protocol could become your new default. As for AT SP h, SP Ah will save the protocol information even if the memory option is off. Note that the 'A' can come before or after the h, so AT SP A3 can also be entered as AT SP 3A.

ST hh [Set Timeout to hh]

After sending a request, the ELM327 waits a preset time before declaring that there was no response from the vehicle (the 'NO DATA' response). Even if there was a response, the ELM327 will wait this time to be sure that there are no more responses coming. The AT ST timeout setting controls the amount of time that the ELM327 waits.

The actual time that the ELM327 waits is about 4 msec x hh, so passing a value of FF results in the maximum time of just over one second. A value of 00 is treated as a special case, setting the timer to the default value of 200 ms.

SW hh [Set Wakeup to hh]

Once a data connection is made with a vehicle, there needs to be data flow every few seconds or the connection will 'go to sleep.' The ELM327 will automatically generate 'wakeup' messages in order to maintain this connection whenever the user is not requesting any data. (The replies to these messages are always ignored, and not seen by the user.)

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

AT Commands (continued)

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

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

TP h [Try Protocol h]

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

TP Ah [Try Protocol h with Auto]

This command is almost the same as the SP Ah one, except that the protocol selected is only tested, and is not immediately saved in the internal (EEPROM) memory. The protocol selected will be tried, and if it fails to initialize, the ELM327 will automatically sequence through all of the protocols, attempting to connect to one of them.

WM xx yy zz aa or **WM xx yy zz aa bb** or
WM xx yy zz aa bb cc [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). The user must provide the three header bytes (xx yy zz), and either one (aa), two (aa bb) or three data bytes (aa bb cc). It is not necessary to provide the checksum byte - the ELM327 creates it for you. The message provided will be periodically sent at the rate determined by the AT SW setting (note that the ELM327 never prints replies to these messages). Byte values assigned with this command are not affected by those set with other commands (AT SH) and do not have any effect on the transmission of normal OBD request messages.

WS [Warm Start]

This command causes the ELM327 to perform a complete software reset very similar to the AT Z command, but not including the power on LED test. Users may find this a convenient means to quickly "start over."

Z [reset all]

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

AT Command Summary

General Commands

D	set all to Defaults
E0	Echo Off
E1	Echo On
I	print the ID
L0	Linefeeds Off (default set by pin 7)
L1	Linefeeds On
WS	Warm Start (quick software restart)
Z	reset all

OBD Commands

AL	Allow Long (>7 byte) messages
BD	perform a Buffer Dump
BI	Bypass the Initialization sequence
DP	Describe the current Protocol
DPN	Describe the Protocol by Number
H0	Headers Off (default)
H1	Headers On
M0	Memory Off (default set by pin 5)
M1	Memory On
MA	Monitor All
MR hh	Monitor for Receiver = hh
MT hh	Monitor for Transmitter = hh
NL	Normal Length (7 byte) messages
PC	Protocol Close
R0	Responses Off
R1	Responses On
SH yzz	Set Header
SH xx yy zz	Set Header
SP h	Set Protocol to h and save it
SP Ah	Set Protocol to Auto, h and save it
ST hh	Set Timeout to hh x 4 msec
TP h	Try Protocol h
TP Ah	Try Protocol h with Auto search

CAN Specific Commands

CAF1	CAN Automatic Formatting On
CAF0	CAN Automatic Formatting Off
CF hhh	set the ID Filter to hhh
CF hh hh hh hh	set the ID Filter to hhhhhhhh
CFC1	CAN Flow Control On
CFC0	CAN Flow Control Off
CM hhh	set the ID Mask to hhh
CM hh hh hh hh	set the ID Mask to hhhhhhhh
CP hh	set CAN Priority (only for 29 bit)
CS	show the CAN Status

ISO Specific Commands

IB 10	set the ISO Baud rate to 10400
IB 96	set the ISO Baud rate to 9600
SW hh	Set Wakeup interval to hh x 20 msec
WM xx yy zz aa	set the Wakeup Message
WM xx yy zz aa bb	" "
WM xx yy zz aa bb cc	" "

Misc. Commands

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

Reading the Battery Voltage

Before proceding to the OBD Commands, we will show an example of how to use an AT Command. We will assume that you have built (or purchased) a circuit which is similar to that of Figure 8 in the Example Applications section. 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
12.6V

>
```

Note that we did not use upper case characters in this example, mostly out of laziness. 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 to the ELM327. Also note that we have shown a space character (' ') between the 'at' and the 'rv'. This is only to separate the commands and make them more readable. You do not have to add spaces, or if you wish, you can add many spaces – it does not affect the internal interpretation of the command.

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 too many changes. Perhaps in this case, you have connected your multimeter, and find that it reads 12.47V, and you would like the ELM327 to read the same. Simply calibrate it to that voltage using the CV command:

```
>at cv 1247
OK
```

At this point, the internal values have been changed, and the ELM327 knows that the current voltage at the input is actually 12.47V. You should not provide the decimal point in the value, as the IC knows that it should be between the second and the third digit. 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. Note that the second decimal place is always maintained internally, and used in the calculations but never displayed.

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

```
>at cv 0932
OK

>
```

Other AT Commands are used in the same manner. Simply type the letters A and T, follow that with the command you want to send, then any arguments that are required for that command, and press return (or enter, depending on your keyboard). You can place 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 received on the RS232 bus 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 single 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 message, and the ELM327 adds these extra bytes to your command bytes automatically. The initial (default) values for these header bytes are usually adequate for most requests, but if you wish to change them, there is a method to do so (see the “Setting the Headers” section).

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

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

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 (decimal value is 166). The header bytes and a checksum byte would be added, and a total of five bytes would typically be sent to the vehicle. Note that the carriage return character is only a signal to the ELM327, and is not sent on to the vehicle.

After sending the command, the ELM327 listens on the OBD bus for messages, looking for ones that are directed to it. If a message address matches, those 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 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 are received, the ELM327 will continue to reset this timer. Note that the IC will always respond with something, even if it is to say “NO DATA” (meaning that there were no messages at all addressed to it).

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 ELM327 cannot be directly connected to a vehicle as it is, but needs support circuitry as shown in the Example Applications section. Once incorporated into such a circuit, one need only use a terminal program to send bytes to and receive them from the vehicle via the ELM327.

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

Normally, one is only concerned with the nine diagnostic test modes described by J1979 (although there is provision for more). All of these modes are not required to be supported by every vehicle, and are often not. These are the nine modes:

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

Within each mode, PID 00 is normally reserved to show which PIDs are supported by that mode. Mode 01, PID 00 must be supported by all vehicles, and can be accessed as follows:

Ensure that the ELM327 is properly connected to your 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 vehicle. At the prompt, issue the mode 01 PID 00 command:

```
>01 00
```

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

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

The 41 00 signifies a response (4) from a mode 1 request from PID 00 (a mode 2, PID 00 request is answered with a 42 00, etc.). The next four bytes (BE, 1F, B8, and 10) represent the requested data, in this case a bit pattern showing the PIDs 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). This is PID 05 in mode 01, and can be requested as follows:

```
>01 05
```

The response will be of the form:

```
41 05 7B
```

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

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

```
>01 0C
```

A typical response would be:

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

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

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

Multiline Responses

There are occasions when a vehicle must respond with more information than one 'message' is able to show. In these cases, it responds with several lines which must be assembled into one complete message.

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

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

Note that all OBD compliant vehicles do not necessarily provide this information. Many older ones do not, but as a rule, the newer ones do. If your vehicle does not support this parameter, you will only see a "NO DATA" response.

The first two bytes (49 and 02) on each line of the above response do not show any vehicle information. They only show that this is a response to an 09 02 request. The next byte on each line shows the order in which the data is to be assembled. Assembling the remainder of the data in that order, and ignoring the first few 00's gives:

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

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

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

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

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

CAN Formatting has been left on (the default), making the reading of the data easier. With formatting on, the sequence numbers are shown with a colon (':') after each, so that they clearly stand out (0:, 1:, etc.). CAN systems add this hex digit (it goes from 0 to F then repeats), to aid in reassembling the data, just as the J1850 vehicle did.

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

The second line shown begins with the familiar 49 02, as this is a response to an 09 02 request. Clearly they are not part of the serial number. CAN will occasionally add a third byte to the response which we see next ('01') showing the number of data items in the response (the vehicle can only have one VIN, so the response says there is only one data item). That third byte can be ignored. This leaves 17 data bytes which are the serial number (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.

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

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

This is difficult to decipher without knowing a little more information. We need to turn the headers on to see 'who' is dong 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) both responding.

Usually the multiline responses are relatively straight-forward to decipher, but they do take some practice. Hopefully this will help to get you going.

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, then, subtract 128 (or 80 hex). When the result is less than 128, simply read the number of stored codes directly.

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

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

Having determined the number of codes stored, the next step is to request the actual trouble codes with a mode 03 request:

```
>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'). As for further examples, if the response had been D016, the code would be interpreted as U1016, while a 1131 would be P1131.

More than one ECU module can respond to requests such as this, so be prepared for the possibility of receiving several lines of response. To determine which ECU is reporting each line would require turning the headers on with the AT H1 command, as discussed in the previous section.

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 test results

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

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

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

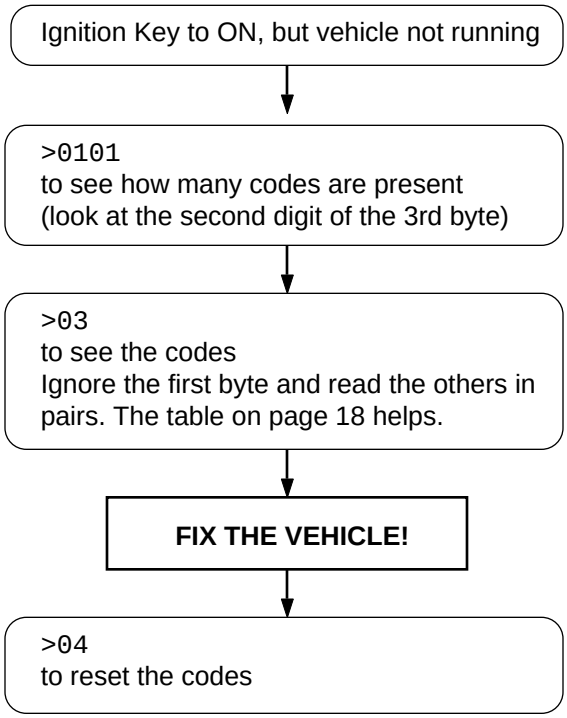
That is all there is to clearing the codes. Once again, be very careful not to accidentally send that 04 code!

Quick Guide for Reading Trouble Codes

If you don't use your ELM327 for some time, this entire data sheet may seem like quite a bit to review when your 'Check Engine' light does eventually come on. 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 be set to either 9600 or 38400 baud, 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 current version of the ELM327 supports several different OBD protocols. As a user, you may never have to choose which one to use, since the factory settings cause an automatic search to be performed for you. While experimenting, you may want the ability to choose, however.

If you know that your vehicle supports a particular protocol, you may want to set the ELM327 to use that protocol only. If, for example, your vehicle is known to use SAE J1850 VPW, and that is all you want, simply look up that protocol in the chart below, then use the 'Set Protocol' AT Command:

```
>AT SP 2
OK
```

From this point on, the default protocol (used after power-up or an AT D command) will be your protocol. Verify this by asking the ELM327 to describe the current protocol:

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

Now what happens if your friend has a vehicle that uses ISO 9141-2? How do you use the ELM327 interface for that vehicle? There are a few choices...

One possibility is to change your protocol selection to allow automatically searching for another protocol, on failure of the current one:

```
>AT SP A2
OK

>AT DP
AUTO, SAE J1850 VPW
```

Now, the ELM327 will always begin by trying protocol 2, but will automatically begin searching for another protocol, should an attempt to connect with protocol 2 fails (as would happen when you connect to a friend's vehicle). Be aware that if you also have the memory function enabled, when you connect to your friend's vehicle, their protocol will be stored in memory as the new default protocol (but it will find yours as the new default when you again connect to your own vehicle).

Perhaps you have disabled the memory function (set pin 5 to 0V), and have used AT SP 2 to customize the IC to your vehicle only. By not using AT SP A2, the interface will not begin searching for another protocol simply because you forgot to turn the ignition key on, which would be an advantage. In this case, you might want to use the 'Try Protocol' command for your friend's vehicle. You can either say:

```
>AT TP 3
OK
```

if you know that your friend's vehicle uses protocol 3, or you can say:

```
>AT TP A3
OK
```

which uses 3 as an initial guess, but then automatically cycles through protocols 1, 2, 3, etc. if the initial one fails to connect.

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

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)

Figure 2. ELM327 Protocol Numbers

OBD Message Formats

To this point we have only discussed the contents of an OBD message, and made only passing mention of other parts such as headers and checksums, which all data packets 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. 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 contents of the message. 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 most 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, and the received data may be falsely interpreted. To detect errors, the various protocols all provide some form of check on the received data, often as simple as a sum calculation (a ‘running total’ is maintained by the receiver as a message is being processed). This is compared to the ‘running total’ sent by the transmitter, and if they do not agree, an error has occured. The total is generally referred to as a ‘checksum’ or a ‘CRC byte’ and is usually sent at the end of a message. If an error is detected, the different protocols provide various ways of handling it.

The OBD data bytes are thus normally encapsulated within a message, with ‘header’ bytes at the beginning, and a ‘checksum’ at the end. The J1850, ISO 9141-2, and ISO 14230-4 protocols all use essentially the same structure, with three header bytes, a maximum of seven data bytes and one checksum byte, as shown in Figure 3 below.

The ISO 15765-4 (CAN) protocol uses a very similar structure, the main difference really only relating to the structure of the header. CAN header bytes are not referred to as that - they are called ‘ID bits’ instead. The initial CAN standard defined the ID bits as being 11 in number, and the more recent CAN

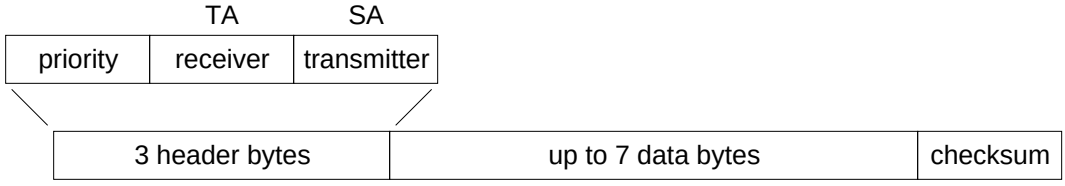


Figure 3. An OBD Message

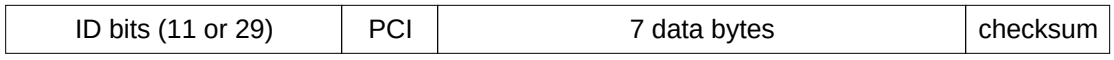


Figure 4. A CAN OBD Message

OBD Message Formats (continued)

standard now allows 29 in total.

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 all three header and the single checksum byte for the J1850, ISO 9141 and ISO 14230 protocols. For the CAN protocols, however, you will instead see the ID bits (sent as hexadecimal digits), and also what is known as a PCI byte, just before the start of the data.

The ELM327 does not display the checksum information for CAN systems, nor does it show the IFR bytes for J1850. If you wish to know more about these last terms, it would be wise to purchase the relevant standard from either ISO, or the SAE.

It is not necessary to ever have to set these header byes or perform one of these checksum calculations – the ELM327 will always do this for you. The header bytes are adjustable however, should you wish to use advanced techniques such as physical addressing.

Restoring Order

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

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

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

```
>AT D
OK
```

This will often be sufficient to restore order, but it can occasionally bring unexpected results. One such surprise will occur if you are connected to a vehicle using one protocol, but the saved default protocol is a different one. In this case, the ELM327 will close the current session and then set 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 two ways that this can be performed with the ELM327. The first is a full hardware reset that acts exactly as if the power were cycled off an then on. It uses the same command as with the our other interface circuits:

```
>AT Z
```

It takes approximately one second for the IC to initialize everything and then perform a test of the four status LEDs, lighting them in sequence. If this is not required, there is a new command that the ELM327 offers. It is the Warm Start command:

```
>AT WS
```

This uses a software reset to perform exactly the same functions as the AT Z, but it does not test the LEDs, so is considerably faster.

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 the 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). Every processor (or ECU) will also respond to what is known as their physical address. It is the physical address that uniquely identifies each module in a vehicle, and permits you to direct more specific queries to only one particular module.

To retrieve information beyond that of the OBDII requirements then, it will be necessary to direct your requests to either a different functional address, or to an ECU's physical address. This is done by changing the data in the message header.

As an example of functional addressing, let us assume that you want to request that the processor responsible for Engine Coolant provide the current Fluid Temperature. You do not know its address, so you consult the SAE J2178 standard and determine that Engine Coolant is functional address 48. J2178 also tells you that for your J1850 VPW vehicle, a priority byte of A8 is appropriate. Then, knowing that a scan tool is normally address F1, you form the information into the three header bytes of A8 48 and F1. To tell the ELM327 to use these new header bytes, all that is needed 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 headers, all one needs to do is issue the secondary ID for fluid temperature (10) at the prompt. If the display of headers is turned off, the conversation could look like this:

```
>10
10 2E
```

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

The other method of obtaining information is by physical addressing, in which you address your request to a specific device, not to a group. To do this, you again need to construct a set of header bytes, that direct the 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 header byte. By monitoring your system for a time with the headers turned on (AT H1), you can quickly learn the main addresses of the senders. When you know the address, simply use it for the second byte in the header. Physical addressing is used by standards such as SAE J2190 to provide a great deal of vehicle information. Many of the details of how to access this information (the PID numbers, etc.) are well-kept secrets that the manufacturers naturally do not wish to share. Elm Electronics does not maintain lists of this information, and can not provide any further details for you.

Advanced experimenters will be aware that the ISO14230 standard also specifies that the first header byte must always include the length of the data field. From that, one might assume that the header would need to be changed for every message. It does not. The ELM327 always determines the number of bytes that you are sending, and inserts that length for you, so you only need to provide the two format bits. Also, we are occasionally questioned about the additional length byte that the generic ISO 14230 standard provides for. The ELM327 only supports that which is required by ISO 14230-4, which is the three byte header, and no additional length byte.

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

The CAN standard states that for diagnostics, the priority byte ('vv' in the diagram) shall always be 1B. Using a separate instruction to set these 'priority' bits should be only a minor inconvenience, as they are rarely changed. The next byte ('xx') describes the type

Setting the Headers (Cont'd)

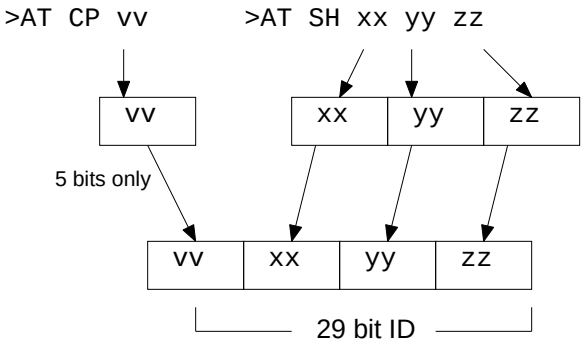


Figure 5. Setting a 29 bit CAN ID

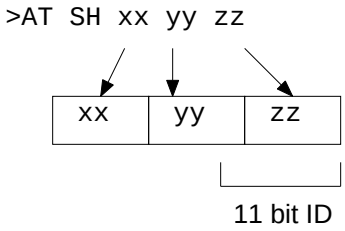


Figure 6. Setting an 11 bit CAN ID

of message that this is, and is set to hex DB for functional addressing, and to DA if using physical addressing. The next two bytes are as defined previously for the other standards - 'yy' is the receiver (or Target Address), and 'zz' is the transmitter (or Source Address). For the functional diagnostic requests, the receiver is always 33, and the transmitter is F1.

Those that are familiar with the SAE J1939 standard will likely find this header structure to be familiar. (J1939 is a CAN standard for use by 'heavy-duty vehicles' such as trucks and buses). We use 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. Although we do not specifically claim to support J1939, we will accept suggestions for improvements that make the ELM327 more useful for use with it. Experimenters are cautioned that auto-formatting (adding PCI bytes) and the sending of flow control messages are on by default in the ELM327. You will need to turn them both off (AT CAF0, and AT CFC0) before sending J1939 messages.

The final header format to discuss is that used in 11 bit CAN systems. They also use a priority/address structure, but shorten it into roughly three nibbles rather than three bytes. The ELM327 uses the same commands to set these values as for other headers, except that it only uses the 11 least significant bits of the provided header bytes, and ignores the others (as shown in Figure 6). It quickly becomes inconvenient to have to enter six digits when only three are required, so there is a special 'short' version of the AT SH command that accepts three hex digits. It actually operates by simply adding zeroes for you.

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

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

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

```
>AT SH 7E0
OK

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

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

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

Monitoring the Bus

Some vehicles use the OBD bus for information transfer during normal vehicle operation, passing a great deal of information over it. A lot can be learned if you have the good fortune to connect to one of these vehicles, and are able to decipher the contents of the messages. Some other vehicles cannot be initialized, and instead continually send information - the only way to read the data from them is by monitoring everything that is being sent, and extracting the useful data.

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

The monitoring mode can be stopped by sending any single RS323 character to the ELM327, or by putting a low logic level on the RTS pin. Any convenient character can be used to interrupt the IC: there are no restrictions on whether it is printable, etc. Note that the character you send will be discarded, and will have no effect on any subsequent commands. The time it takes to respond to this interrupting character will depend on what the ELM327 is doing when it is received. The IC will always finish a task that is in progress (printing a line, for example) before returning to wait for input, so you should always wait for the prompt character ('>'), or the Busy line to go low, before beginning to send a command.

One unexpected result may occur if you have the 'Auto' protocol search feature enabled, and you tell the ELM327 to begin monitoring when a bus is quiet. In this case, the ELM327 will begin searching for another protocol, which can be unexpected. Be aware also that the ISO 9141 and ISO 14230 protocols look identical when monitoring, so the ELM327 will likely stop searching at ISO 9141, even if the protocol is ISO 14230. With the Automatic searching enabled, this should correct itself, however, when the first OBD request is made.

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

```
>AT MT 10
```

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

Using this command with 11 bit CAN systems can be a little confusing at first. Recall the way in which all header bytes are stored within the ELM327. An 11 bit CAN ID is actually stored as the least significant 11 bits in the 3 byte 'header storage' location. It will be stored with 3 bits in the receiver's address location, and the remaining 8 bits in the transmitter's address location. For this example, we have requested that all messages created by transmitter '10' be printed, so all 11 bit CAN IDs that end in 10 will be displayed (ie '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 specific 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 need some further explanation. It may be helpful to first picture the hex number '10' as the binary number '0001 0000'. This is the number that the ELM327 would normally use to match to the second byte of incoming messages. Also, recall that all of the 11 bit CAN IDs are stored in the header storage, with the data 'right justified.' The ELM327 expects this, and only ever uses 3 bits from the second header byte and 8 bits from the third byte for all 11 bit CAN messages. The rest are ignored.

In this case then, the ELM327 is provided with a byte to match in the second position, but it only looks at the three rightmost bits, which are all zeros (000). All messages that begin with '0' as the first digit will actually be displayed if you say AT MR 10. To search for all CAN messages that begin with a 2, then you will need to use the command 'AT MR 02', and to see all of the 7xx's, you will need to use 'AT MR 07'.

CAN Messages and Filtering

The ELM327 monitoring commands (AT MA, MR and MT) usually work very well with the 'slower' protocols – J1850, ISO 9141 and ISO 14230. The CAN systems are a different story, however, as they often have an order of magnitude more information passing over them. The relatively small 256 byte buffer that the ELM327 uses for sending can quickly fill up when data is arriving at 500 Kbps and leaving at 38.4 Kbps.

To help reduce the amount of information seen by the ELM327, the internal CAN module has a 'filter' that can be used to pass only messages with specific ID bits. A range of values can be passed when the filter is used with what is called a 'mask' to say which bits are relevant.

As an example, consider an application where you are trying to monitor for 29 bit CAN diagnostic messages, exactly like the ELM327 does. By definition, these messages will be sent to the scan tool at address F1. From ISO 15765-4, you know then that the ID portion of the reply must be of the form:

18 DA F1 xx

where xx is the address of the module that is sending the message. To use the filter, then, enter what you have into it, putting anything in for the unknown portion (you will see why in a moment). The command to set the CAN filter is AT CF…

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

How, you ask, do you tell the ELM327 to ignore those last two 0's? You do that with the mask. The mask is a set of bits that tell the ELM327 which bits in the filter are relevant. If the mask bit is 1, that filter bit is relevant, and is required to match. If it is 0, then that filter bit will be ignored. All bits in the above message are relevant, except those of the last two digits. To set the mask for this example then, you would need to use the CAN Mask command, as follows:

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

If desired, you can convert the hexadecimal to binary to see what has been done.

The 11 bit CAN IDs are treated in the same manner. Recall that they are stored internally in the right-most 11 bits of the locations used for 29 bit CAN, which must be considered when creating a filter or mask. As an example, assume that we wish to display all messages that have a 6 as the first digit of the 11 bit ID. We need to set a filter to look for 6 in that digit:

```
>AT CF 00 00 06 00
```

The 11 bit ID is stored in the last three locations, so the 6 would appear where it is shown. Now, to make that digit relevant, we create the mask:

```
>AT CM 00 00 0F 00
```

The system only uses the 11 right-most bits in this case, so we can be lazy and enter the F as shown (the first bit of the F will be ignored, and it will be treated as if we had entered a 7).

Clearly, this can be quite cumbersome if using 11 bit CAN systems routinely. To help with that, the ELM327 offers some shorter versions of the CF and CM commands. You need only enter:

```
>AT CF 600
```

and

```
>AT CM F00
```

for the above example. The commands work internally by simply entering the extra 00's for you. As for the full eight digit versions, only the 11 least significant (right-most) digits are used, so you do not need to take special care with the first bit.

With a little practice, these commands are fairly easy to master. Initially, try entering the filter and mask values, then use a command such as AT MA to see what the results are. The ELM327 knows that you are trying to filter, and combines the effects of both commands (it will do that for MR and MT as well). The MA, MR and MT commands also have the extra benefit that if they are in effect, the ELM327 will remain quiet, not sending acknowledgement or error signals, so anything you do while monitoring should not disrupt others that are on the bus.

Note that if a filter has been set, it will be used for all CAN messages, so standard OBD requests may then respond with "NO DATA". If you are having trouble, reset everything to the default values.

CAN Message Formats

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

SF - the Single Frame

FF - the First Frame (of a multiframe message)

CF - the Consecutive Frame (“ “)

FC - the Flow Control frame

The Single Frame message contains storage for up to seven data bytes and what is known as a PCI (Protocol Control Information) byte. The PCI byte is always the first byte of them all, and tells how many data bytes are to follow. If the CAN Auto Formatting option is on (CAF1) then the ELM327 will create this byte for you when sending, and remove it when receiving (but if the headers are enabled, you will always see it).

If you turn the Auto Formatting off (with CAF0), it is expected that you will provide all of the data bytes to be sent. For diagnostics systems, this means the PCI byte and the data bytes. The ELM327 will add extra padding bytes for you (value ‘00’) however, to ensure that you send eight data bytes (as that parameter is not adjustable with this version of the ELM327). You do not need to set the Allow Long (AT AL) option in order to do this, as the IC overrides it for you.

A First Frame message is used to say that a multiframe message is about to be sent, and tells the receiver just how many data bytes to expect. The length descripter is limited to 12 bits, so a maximum of 4095 byes can be received at once using this method.

Consecutive Frame messages are sent after the First Frame message to provide the remainder of the data. Each Consecutive Frame message includes a single hex digit ‘sequence number’ that is used to help with reassembling the data. It is expected that if a message were corrupted and resent, it could be out of order by a few packets, but not by more than 16. As seen previously, the serial number for a vehicle is often a multiframe response:

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

In this example, the line that begins with 0: is the First Frame message. The length (014) was actually extracted from the message by the ELM327 and printed on the separate line as shown. Following the First Frame line are two Consecutive Frames as shown (1: and 2:). To learn more details of the exact formatting, you may want to send a request such as the one above, then repeat the same request with the headers enabled (AT H1). This will show the PCI bytes that are actually used to send these components of the total message.

The Flow Control frame is one that you do not normally have to deal with. When a First Frame message is sent as part of a reply, the ELM327 must tell the sender some technical things such as how long to delay between Consecutive Frames, etc. These are predefined by the ISO 15765-4 standard and are not alterable by the user. The only thing that you can do with them is to disable the sending of Flow Control messages entirely (AT CFC0). This may be required if experimenting with a different CAN system.

If a Flow Control frame is received while monitoring, the line will be prefixed with a ‘FC: ’ before the data is displayed, to help with decoding 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 looks for this flag, or for zero data bytes and may report to you that an RTR was detected. 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, RTS may be shown.

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 set to an incorrect baud rate). When erorrs do occur, the ELM327 will print all bytes (no matter what CAF, etc., is set to), followed by the message ‘<RX ERROR’.

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) process, while ISO 14230 allows for both the slow method, and a faster alternative. In either case, once the bus has been initiated, communications must take place at least once every five seconds, or the bus will revert to a low-power 'sleep' mode.

The ELM327 takes care of this bus initiation and the periodic sending of 'keep-alive' or 'wakeup' messages for you – it is automatic and requires no input from the user. The ELM327 will not perform the bus initiation until the first message needs to be sent, however. During the automatic search process, you will not see any status reporting while the initiation process is taking place, but if you have the Auto option off, then you will see a message similar to this:

BUS INIT: ...

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

Once initiated, the ELM327 does what is required to keep the bus alive, without any intervention from the user. If you have installed monitoring LEDs, you will be able to see that automatic messages being sent every few seconds, if there is no other bus activity.

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. Users generally do not have to change either of the above though, as the default settings work with almost all systems.

Wakeup Messages

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

We term these periodic messages that are created the 'Wakeup Messages.' They keep the connection alive, and prevent the circuitry from going back to the idle or sleep mode. Some texts refer to these messages simply as 'idle messages.' The ELM327 automatically creates and sends these for you if there appears to be no other activity – there is nothing that you need do to ensure that they occur. To see these, once a connection is made, simply monitor the OBD transmit LED - you will see the periodic 'blips' created when the ELM327 sends one. If you are curious as to the actual contents of the messages, you can then perform a Buffer Dump to see the bytes. Note that the ELM327 never prints a response to any of these wakeup messages.

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

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

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

From that point forward, every wakeup message that the ELM327 sends will be as shown above.

You can change these as often as you want, the only restriction being that every time you do, you must provide the complete message - three header bytes followed by either one, two, or three data bytes. You need not worry about providing a checksum, as it will be added for you.

Error Messages

When hardware or data problems occur, the ELM327 will respond with one of the following short messages. Some of the messages are suppressed during an automatic search for a protocol, and are only visible if not in the “Auto” mode. Here is a brief description of each:

BUFFER FULL

The ELM327 provides a 256 byte internal RS232 transmit buffer so that OBD messages can be received quickly, stored, and sent to the computer or PDA at a more constant rate. Occasionally (particularly with CAN systems) the buffer will fill at a faster rate than it is being ‘emptied.’ Eventually it becomes full, and no more data can be stored (it is lost).

If you are receiving BUFFER FULL messages, and are using a 9600 baud data rate, give serious consideration to changing your data rate to 38400 baud. If you still receive BUFFER FULL messages after that, consider some of the filtering options (the MR, MT, CF and CM AT Commands).

BUS BUSY

The ELM327 tried to send the mode command or initialize the bus, but detected too much activity to insert a message. This could be because the bus was in fact busy, but may be due a wiring problem that is giving a continuously active input. If this is an initial trial with your ELM327, check the voltage levels at your OBD input – this is very likely a wiring problem.

BUS ERROR

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

CAN ERROR

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

FB ERROR

When an OBD output is energized, a check is made to ensure that the signal appears at the respective input. If there is a problem, 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 very likely a wiring problem. Check your wiring before proceeding.

DATA ERROR

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

<DATA ERROR

The was an error in the line pointed to, either from an incorrect checksum or a problem with the format of the message (the ELM327 still shows you what it received). There could have been a noise burst which interfered, or a circuit problem. Try re-sending the command.

NO DATA

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

<RX ERROR

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

UNABLE TO CONNECT

The ELM327 has tried all of the available protocols, and could not detect a compatible one. This could be because your vehicle uses an unsupported protocol, or could be as simple as forgetting to turn the ignition key on. Check all of your connections, and the ignition, then try the command again.

?

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

Computer Control

A common question we receive with our OBD interpreters is “Can I connect my ELM32x circuit directly to my own circuit, or must I use the RS232 interface shown?” Certainly you may connect directly to it, you do not need to use RS232 voltage levels. The ELM327 is a CMOS device that uses industry standard levels for all inputs and outputs. The outputs are actually able to drive several mA without problem, so can directly drive LEDs, etc.

With the ELM327, we have stopped using an inverted RS232 Rx input, so interfacing is even simpler. If you have a microprocessor that uses the same 5V supply as the ELM327, and has an internal UART, all you will generally need do is connect your processor’s transmit output to the ELM327’s receive input, and your receive input to the ELM327’s transmit output. Communications should work with this connection (and a correct baud rate setting).

The ELM327 has a new hand-shaking feature that may be very useful for many interfaces. There is a request to send (RTS) input and a Busy output. To use them, set one of your port pins to normally provide a high output, and connect it to the RTS input. Use another port pin as an input to monitor the ELM327 Busy output. When you want to send a command, check the Busy output first. If it is at a logic high, then bring your RTS line low and wait for the Busy line to go low. When it does, restore your RTS line to a high level, and then send your command to the ELM327. Not to worry about the ELM327 becoming Busy again after you raise the RTS line - once Busy goes low, the ELM327 waits for your command.

Example Applications

The SAE J1962 standard dictates that all OBD compliant vehicles must provide a standard connector near the driver’s seat, the shape and pinout of which is shown in Figure 7 below. The circuitry described here can be used to connect to this J1962 plug without modification to your vehicle.

The male J1962 connector required to mate with a

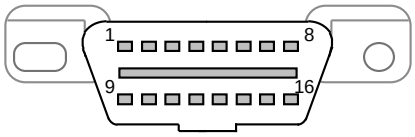


Figure 7. The J1962 Vehicle Connector

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

The circuit of Figure 8 on page 32 shows how the ELM327 might typically be used. Circuit power is obtained from the vehicle (via OBD pins 16 and 5) and, after a protecting diode and some capacitive filtering, is presented to a five volt regulator. (Note that a few vehicles have been reported to not have a pin 5 – on these you will use pin 4 instead of pin 5.) The regulator powers several points in the circuit as well as an LED (for visual confirmation that power is present). We have shown a 78L05 for the regulator as that limits the current available to about 100mA which is a safe value for experimenting. The CAN interface is a low impedance circuit however, and if doing sustained transmissions on CAN, this type of regulator may shut down on over-temperature. Should you experience this problem, you may want to consider a 1 Amp version.

Shown in the top left corner is the CAN interface circuitry. It is strongly recommended that you implement something similar to what we have shown,

Example Applications (Cont'd)

using a commercial transceiver chip. Some CAN buses have a lot of critical information on them, and mostly for reasons of data integrity, it is not advised that you develop your own discrete interface. The Microchip MCP2551 is shown here, but many other manufacturers produce CAN transceiver ICs, such as Philips (82C251), Texas Instruments (SN65LBC031), and Linear Technology (LT1796). Generally these are very low-cost devices, that can save a lot of grief.

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

The ELM327 controls both of the ISO lines through the NPN transistors shown, with the pullup resistors connected to their collectors. The 510Ω value for these resistors is specified in the standards, but you can use a close value if required (people often write us concerning this). If you have to, you can likely go to 560Ω for these resistors without experiencing ill effects. Note that reducing the value could cause circuit damage, so that should be avoided. Try to keep as close as possible to the 510Ω. Note also that 1/2W resistors should be used (and that 1/4W 240Ω + 270Ω resistors work well, too).

Data is received from the K Line of the OBD bus and connected to pin 12 after being reduced by the R20/R21 voltage divider shown.

The final OBD interface shown is for the two J1850 standards. The J1850 VPW standard needs a positive supply of up to 8V while the J1850 PWM needs 5V. This dual voltage supply is provided by the 317L adjustable regulator shown controlled by pin 3. With the resistor values given, the voltages are switched between about 7.5V and 5V, which works well. The two outputs are driven by the Q1 and Q2 combination for the Bus +, and Q3 for the Bus -.

The voltage monitoring circuitry for the AT RV command is shown in this schematic connected to pin 2 of the ELM327. The two resistors simply divide the battery voltage to a safe level for the ELM327, and the capacitor filters out noise. As shipped, the ELM327 expects a resistor divider ratio as shown, and sets nominal calibration constants assuming that. If your application needs a different range of values, simply adjust the resistor values, then perform an AT CV to calibrate to that (but the ELM327 can not display more than 99.9V).

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

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

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

We often receive requests for parts lists to accompany our Example Applications circuits. Since this circuit is more complex than most, we have named/numbered all of the components and provided a summary parts list (see Figure 9 on page 33). Note that these are only suggestions for parts. If you prefer another LED colour, or have a different general purpose transistor on hand, etc., by all means make the change. A quick tip for those having trouble finding a 0.3" wide socket for the ELM327: many of the standard 14 pin sockets can be placed end-to-end to form one 0.3" wide 28 pin socket.

The ELM327 was built to be a multi-protocol device that automatically searches for a valid protocol, but there is no reason that it can not be used in a circuit that supports only one protocol. Figure 10 on page 34 shows an example of how the ELM327 might be used in a 'J1850 VPW only' circuit.

The differences between Figures 8 and 10 should be apparent. Unused protocols have simply had their outputs ignored (left open circuit), and their inputs wired to a convenient logic level. Note that these are CMOS inputs, so must never be left floating.

The circuit maintains the voltage measuring input circuitry, the status LEDs, and the J1850 Bus+ circuitry, but the majority of the rest has been eliminated. The voltage switching circuitry has been

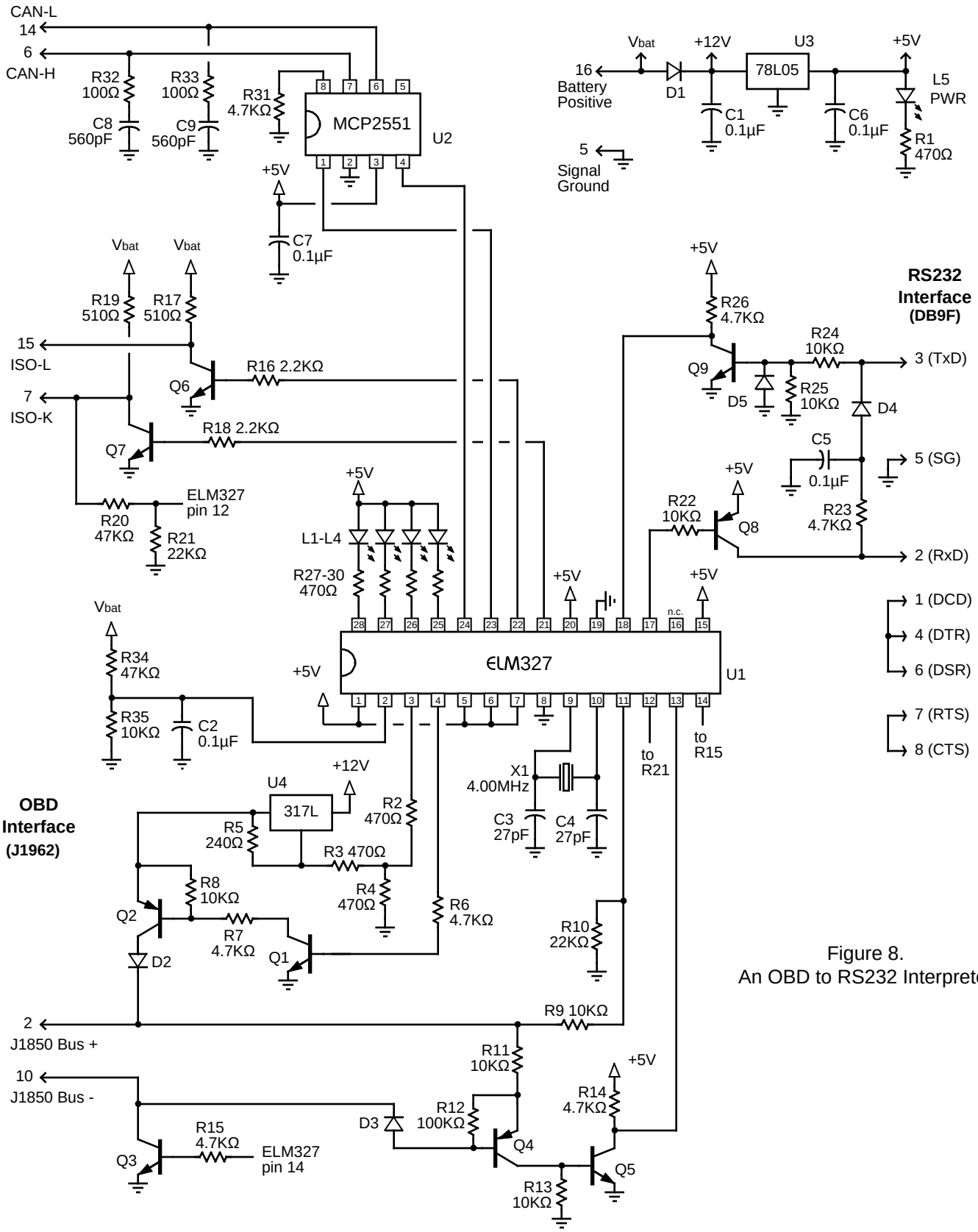


Figure 8.
An OBD to RS232 Interpreter

Example Applications (Cont'd)

reduced to a single 8V regulator as well, since there will be no need to switch to 5V.

The first time that this circuit is used, it will likely be set to protocol 0 – the default 'Automatic search' mode of operation (as shipped from the factory). When you connect it to the J1850 VPW vehicle, it will then first try a J1850 PWM (protocol 1) connection, fail, and proceed to try J1850 VPW. If the memory is enabled (as shown), J1850 VPW will then become the new default. This will work well for most applications, but if the circuit is used on a vehicle with the key off, for example, then it will again go searching for a new protocol.

In general, you do not want this to happen every time. It may be only a minor inconvenience to have to wait while the ELM327 determines that it is "UNABLE TO CONNECT", but why go through it if you do not have to? If you know that you are using the circuit in a J1850 VPW only application (protocol 2) then you should issue the command AT SP 2 the very first time that the circuit is powered. From that point on, it will remain in protocol 2, whether it fails to make a connection or not.

That provides two examples of how this integrated circuit might be used. Certainly, it is not restricted to only those, but hopefully it is enough to get you started…

Semiconductors
D1 = 1N4001
D2, D3, D4, D5 = 1N4148
L1, L2, L3, L4 = Yellow LED
L5 = Green LED
Q1, Q3, Q5, Q6, Q7, Q9 = 2N3904 (NPN)
Q2, Q4, Q8 = 2N3906 (PNP)
U1 = ELM327
U2 = MCP2551
U3 = 78L05 (5V, 100mA regulator)
U4 = 317L (adj. 100mA regulator)

Capacitors
C1, C2, C5, C6, C7 = 0.1uF 16V
C3, C4 = 27pF
C8, C9 = 560pF

Resistors
R32, R33= 100 Ω
R5 = 240 Ω
R1, R2, R3, R4, R27, R28, R29, R30 = 470 Ω
R17, R19 = 510 Ω 1/2W
R16, R18 = 2.2 KΩ
R6, R7, R14, R15, R23, R26, R31 = 4.7 KΩ
R8, R9, R11, R13, R22, R24, R25, R35 = 10 KΩ
R10, R21 = 22 KΩ
R20, R34 = 47 KΩ
R12 = 100 KΩ

Misc
X1 = 4.000MHz crystal
RS232 Conn = DB9F
IC Socket = 28pin 0.3" (or 2 x 14pin)

Figure 9. Parts List for Figure 8

Example Applications (Cont'd)

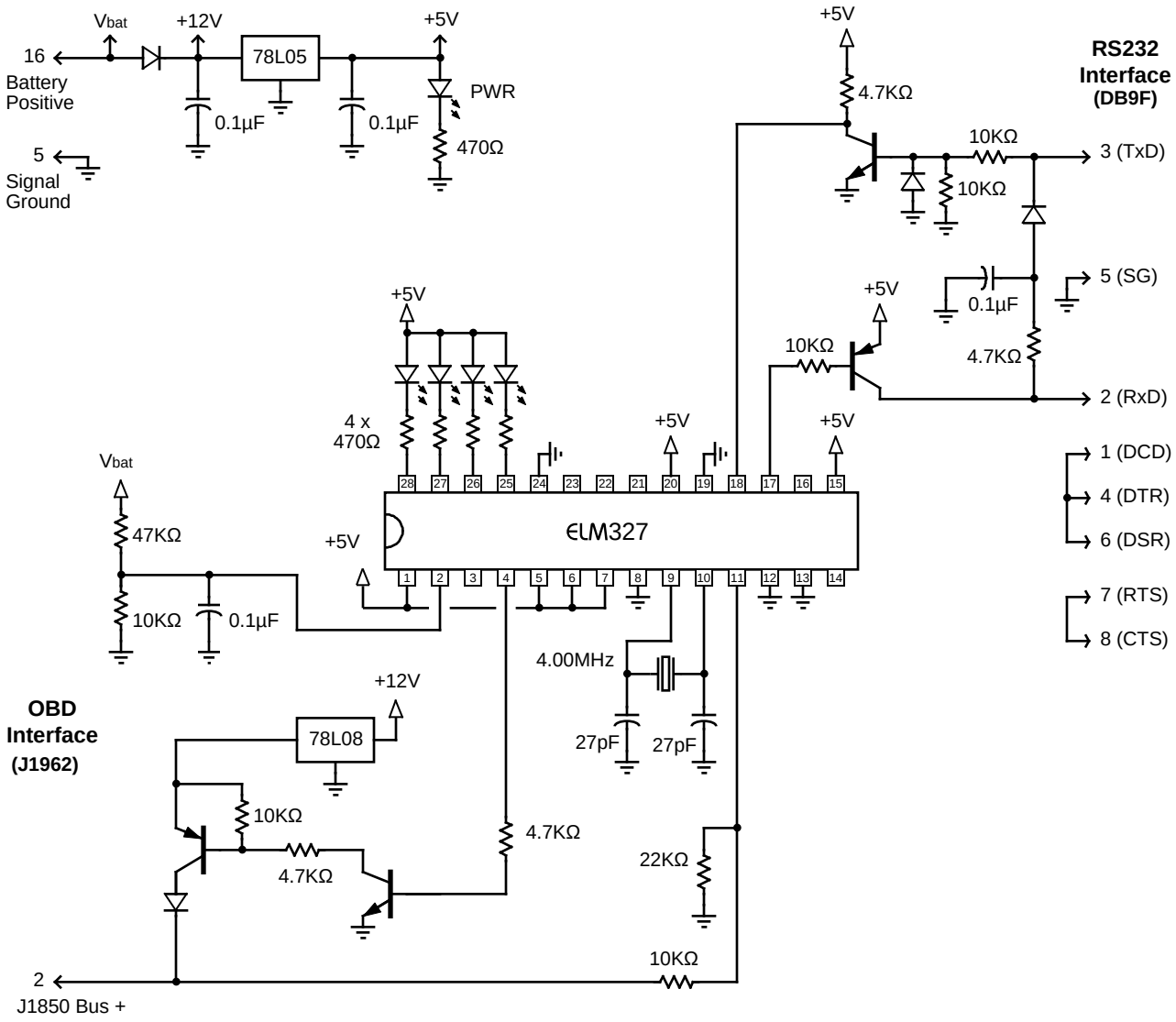


Figure 10. An OBD (J1850 VPW) to RS232 Interpreter

Section Index