ELM322

OBD (VPW) to RS232 Interpreter

Description

Since the 1996 model year, North American automobiles have been required to provide an OBD, or On Board Diagnostics, port for the connection of test equipment. Data is transferred serially between the vehicle and the external equipment using this connection, in a manner specified by the Society of Automotive Engineers (SAE) standards. In addition to operating at different voltage levels, these ports also use a data format that is not compatible with the standard used for personal computers.

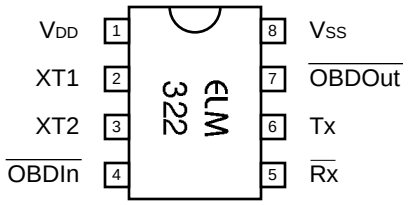
The ELM322 is an 8 pin integrated circuit that is able to change the data rate and reformat the OBD signals into easily recognized ASCII characters. This allows virtually any personal computer to communicate with an OBD equipped vehicle using only a standard serial port and a terminal program. By also enhancing it with an interface program, hobbyists can create their own custom scan tool.

This integrated circuit was designed to provide a cost-effective way for experimenters to work with an OBD system, so a few features such as RS232 handshaking, variable baud rates, etc., have not been implemented. In addition, this device is only able to communicate using the 10.4KHz J1850 VPW protocol that is commonly used in General Motors and some Daimler Chrysler vehicles.

Features

- Low power CMOS design
- High current drive outputs
- Crystal controlled for accuracy
- Fully configurable using AT commands
- Standard ASCII character output
- High speed RS232 communications
- 10.4 KHz J1850 VPW protocol

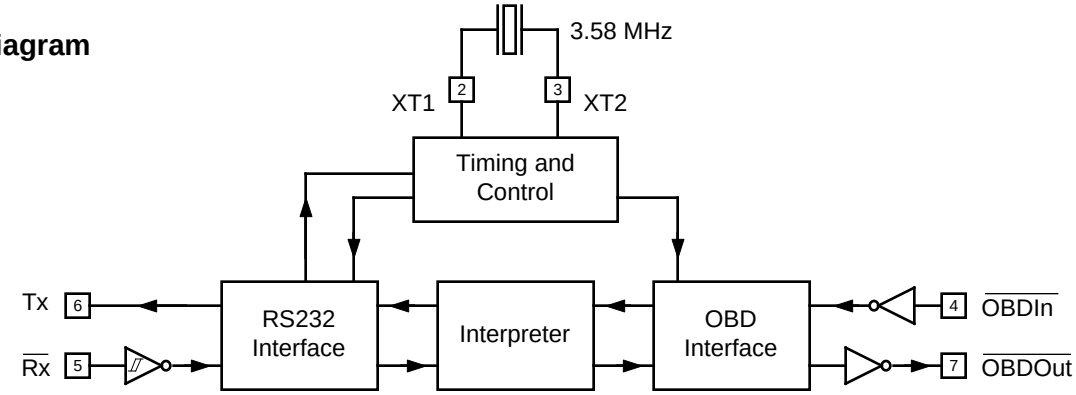
Connection Diagram
PDIP and SOIC
(top view)



Applications

- Diagnostic trouble code readers
- Automotive scan tools

Block Diagram



Pin Descriptions

V_{DD} (pin 1)

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.

XT1 (pin 2) and XT2 (pin 3)

A 3.579545 MHz NTSC television colourburst crystal is connected between these two pins. Crystal loading capacitors (typically 27pF) will also normally be connected between each of the pins and the circuit common (Vss).

$\overline{\text{OBDIn}}$ (pin 4)

The OBD data is input to this pin, with a low logic level representing the active state (and a high, the passive). No Schmitt trigger input is provided, so the OBD signal should be buffered to minimize transition times for the internal CMOS circuitry. The external level shifting circuitry is usually sufficient to accomplish this - refer to the Example Applications section for a typical circuit.

$\overline{\text{Rx}}$ (pin 5)

The computer's RS232 transmit signal can be directly connected to this pin from the RS232 line, as long as a current limiting resistor (typically about 47KΩ) is installed in series. (Internal protection diodes will pass the input currents safely to the supply connections, protecting the ELM322.) Internal signal inversion and Schmitt trigger waveshaping provide the necessary signal conditioning.

Tx (pin 6)

The RS232 data output pin. The signal level is compatible with most interface ICs, and there is sufficient current drive to allow interfacing using only a single PNP transistor, if desired.

$\overline{\text{OBDOut}}$ (pin 7)

This is the active low output signal which is used to drive the OBD bus to the active (high) state. Typically this is accomplished by switching a PNP type transistor on - see the Example Applications section for more details.

V_{SS} (pin 8)

Circuit common is connected to this pin. This is the most negative point in the circuit.

Ordering Information

These integrated circuits are available in either the 300 mil plastic DIP format, or in the 208 mil SOIC surface mount type of package. To order, add the appropriate suffix to the part number:

300 mil Plastic DIP..... ELM322P 208 mil SOIC..... ELM322SM

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}..... -0.6V to (V_{DD} + 0.6V)

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}		1.0	2.4	mA	see note 3
Input low voltage	V_{SS}		0.15 V_{DD}	V	
Input high voltage	0.85 V_{DD}		V_{DD}	V	
Output low voltage			0.6	V	Current (sink) = 8.7 mA
Output high voltage	V_{DD} - 0.7			V	Current (source) = 5.4 mA
$\overline{Rx}$ pin input current	-0.5		+0.5	mA	see note 4
RS232 baud rate		9600		baud	see note 5

Notes:

1. This integrated circuit is produced with a Microchip Technology Inc.'s PIC12C509A 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. This specification represents the current flowing through the internal protection diodes when the voltage connected to the $\overline{Rx}$ input (through a current limiting resistance) is greater than V_{DD} or less than V_{SS}. Currents quoted are the maximum that should be allowed to flow continuously.
5. Nominal data transfer rate when a 3.58 MHz crystal is used as the frequency reference. Data is transferred to and from the ELM322 with 8 data bits, no parity, and 1 stop bit (8 N 1).

Overview

The following describes how to use the ELM322 to obtain a great deal of information from your vehicle. To some, the quantity of information will be overwhelming, and for others it will not be enough.

We begin by discussing just how to talk to the IC, then how to adjust some options through the use of 'AT' commands, and finally go on to actually talk to the vehicle, obtaining trouble codes and resetting them. For the more advanced experimenters, there are also sections on how to use some of the programmable features of this product as well.

It is not as daunting as it first appears. 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 provide in the following…

Communicating with the ELM322

The ELM322 relies on a standard RS232 type serial connection to communicate with the user. The data rate is fixed at 9600 baud, with 8 data bits, no parity bit, 1 stop bit, and no handshaking (often referred to as 9600 8N1). All responses from the IC are terminated with a single carriage return character and, by default, a line feed character as well. Make sure your software is configured properly for the mode you have chosen.

Properly connected and powered, the ELM322 will initially display the message:

```
ELM322 v2.0

>
```

In addition to identifying the version of the IC, receipt of this string is a convenient way to be sure that the computer connections and the settings are correct. However, at this point no communications have taken place with the vehicle, so the state of the OBD connection is still unknown.

The '>' character displayed above is the ELM322's prompt character. It indicates that the device is in its idle state, ready to receive characters on the RS232 port. Characters sent from the computer can either be intended for the ELM322's internal use, or for reformatting and passing on to the vehicle's OBD bus.

Commands for the ELM322 are distinguished from those to the vehicle by always beginning with the characters 'AT' (as is common with modems), while commands for the OBD bus can contain only the ASCII characters for hexadecimal digits (0 to 9 and A to F). This allows the ELM322 to quickly determine where the received characters are to be directed.

Whether an 'AT' type internal command or a hex string for the OBD bus, all messages to the ELM322 must be terminated with a carriage return character (hex '0D') before they will be acted upon. If an incomplete string is sent and no carriage return appears, an internal timer will automatically abort the incomplete message after about 10 seconds. Should this happen, the ELM322 will print a single question mark to show that the input was not understood (and was ignored).

Messages that are misunderstood by the ELM322 (syntax errors) will always be signalled by a single question mark ('?'). These include incomplete messages, invalid hexadecimal digit strings, or incorrect AT commands. It is not an indication of whether or not the message was understood by the vehicle. (The ELM322 is a protocol interpreter that makes no attempt to assess 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, so it cannot determine if the message sent to the vehicle is in error.)

Incomplete or misunderstood messages can also occur if the controlling computer attempts to write to the ELM322 before it is ready to accept the next command (as there are no handshaking signals to control the data flow). To avoid a data overrun, users should always wait for the prompt character ('>') before issuing the next command.

Finally, a few convenience items to note. The ELM322 is not case-sensitive, so 'ATZ' is equivalent to 'atz', and to 'AtZ'. The device ignores space characters as well as control characters (tab, linefeed, etc.) in the input, so they can be inserted anywhere to improve readability, and finally, issuing only a single carriage return character will repeat the last command (making it easier to request updates on dynamic data such as engine rpm).

AT Commands

Several parameters within the ELM322 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 the settings, for example by turning the character echo off, adjusting the timeout value, or changing the header addresses. 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 ELM322 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 ELM322 will reply with the characters 'OK' on the successful completion of a command, so the user knows that it has been executed.

Some of the following commands allow the passing of numbers as arguments in order to set the internal values. These numbers will always be in hexadecimal format, and must be provided in pairs. The hexadecimal conversion chart in the next 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 will be either the number 1 or 0, the universal terms for on and off.

The following is a summary of all of the AT commands that are recognized by the current version of the ELM322, sorted alphabetically. Users of previous versions of this product (v1.x) should note that their ICs will only support the E, H and Z options.

AR [Automatically set the Receive address]

Responses from the vehicle will be acknowledged and displayed by the ELM322, if its internally stored receive address matches the address that the message is being sent to. With the Auto Receive mode in effect, the value used for the receive address will be chosen based on the current header bytes, and will automatically be updated whenever the header bytes are changed.

The value that is used for the receive address is determined based on the contents of the first header byte. If it shows that the message uses physical addressing, the third byte of the header is used for the receive address, otherwise (for functional addressing) the second header byte, increased in value by 1, will be used. Auto Receive is turned on by default.

D [set all to Defaults]

This command is used to set the E, H, L, and R options to their default (or factory) settings, as when power is first applied. Additionally, the Auto Receive mode (AR) will be selected, data will be transmitted in the standard formatted way (as if chosen by FD), the 'NO DATA' timeout will be set to its default value, and the header bytes will be set to the proper values for OBDII (SAE J1979) operation.

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).

FD [send Formatted Data]

This command requests that all responses be returned as standard ASCII characters, which are readable with virtually any terminal program. Hex digits are shown as two ASCII characters, and spaces are provided between each byte as a separator. Also, every line will end with a carriage return character and (optionally) a linefeed character, ensuring that every response appears on a new line. This is the default mode.

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

These commands control whether or not the header information is shown in the responses. All OBD messages have an initial (header) string of three bytes and a trailing check digit which are normally not displayed by the ELM322. To see this extra information, users can turn the headers on by issuing an ATH1. The default is H0 (headers off).

AT Commands (continued)

I [Identify yourself]

Issuing this command causes the chip to identify itself, by printing the startup product ID string (this is currently 'ELM322 v2.0'). Software can use this to determine exactly which integrated circuit it is talking to, without resorting to resetting the entire IC.

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

Whether or not the ELM322 transmits a linefeed character after each carriage return character is controlled by this option. If an ATL1 is issued, linefeed generation will be turned on, and for ATL0, it will be off. Users may wish to have this option on if using a terminal program, but off if using a custom interface (as the extra characters transmitted will only serve to slow the vehicle polling down). The default setting is L1 (linefeeds on).

MA [Monitor All messages]

Using this command places the ELM322 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. To stop the monitoring, one should send any single character then wait for the ELM322 to respond with a prompt character ('>'). 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 was simply waiting for input, it would return immediately. 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 (messages which are found to have that value in their second byte). Any RS232 activity (single character) aborts the monitoring, as with the MA command.

MT hh [Monitor for Transmitter hh]

Another monitoring command, MT hh displays only messages sent by Transmitter address hh (given by the third byte in the message). As with the MA and MR monitoring modes, any RS232 activity (single character) will abort the monitoring.

PD [send Packed Data]

This option is for those who are building a computer interface and want the fastest data transfer rate possible while still operating at 9600 baud. When selected, responses from the vehicle will be formatted as an initial length byte followed by the actual response bytes from the vehicle, with no trailing carriage returns or linefeed characters. The data will not be altered in any way, except for the conversion to standard RS232 bytes.

Note that the length byte only represents the total number of data bytes following, and does not include itself. Also, if there was a data (checksum) error, the length byte will have its most significant bit set, so the user should always check first to see if the length is greater than 127. (The other 7 bits still provide a valid byte count if there is an error, so one need only ignore the msb, or subtract 128 from the value.)

A 'NO DATA' response has no data bytes, but still sends a length byte with value '0'.

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

These commands control the ELM322's automatic display of responses. If responses have been turned off, the IC will not wait for anything to be returned from the vehicle after sending a request, and will return immediately to waiting for RS232 commands. 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. The default is R1 (responses on).

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

This command allows the user to control the values that are sent as the three header bytes in the message. The value of hex digits xx will be used for the first or priority/type byte, yy will be used for the second or target byte, and zz will be used for the third or source byte. These remain in effect until set again, or until restored to the default values with the AT D, or AT Z commands. The default header values are 68 6A F1, as required by the SAE J1979 Diagnostic Test Modes (OBDII) standard.

AT Commands (continued)

SR hh [Set the Receive address to hh]

Depending on the application, users may wish to manually set the address to which the ELM322 will respond. Issuing this command will turn off the AR mode, and force the IC to only accept responses addressed to hh. This could be useful in non-OBD applications, or simply while experimenting with a network.

ST hh [Set Timeout to hh]

After sending a request, the ELM322 waits a preset time before declaring that there was no response from the vehicle (the 'NO DATA' response). Depending on the application (and priority of the request), users may want to modify this timeout period before the ELM322 declares that the request was a failure. The ST command is used to do that.

The actual time used is (approximately) 4 ms x the byte value passed as the hexadecimal argument. Passing a value of FF thus results in a maximum time of about 1020 ms. Values less than 08 will be ignored and forced to a value of 8, providing a minimum time of 32 ms. The default value is 32 (decimal 50) providing a timeout of 200 ms.

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

Figure 1 (at the right) shows all of the ELM322 commands in one convenient chart. In order to help with the understanding of these, we have grouped the commands into three functional areas, but this has no bearing on how the commands should be used, it is only for clarity. You may find this chart to be useful when experimenting with the IC.

ELM322 AT Commands

general
- **D** set all to Defaults
- **I** show the ID string
- **Z** reset all
- **<CR>** repeat last command

responses
- **E1/0** Echo on/off
- **H1/0** Headers on/off
- **L1/0** Linefeeds on/off
- **R1/0** Responses on/off
- **PD** use Packed Data
- **FD** use Formatted Data
- **ST hh** Set Timeout (hh*4ms)

requests
- **SH xx yy zz** Set Header
- **SR hh** Set Rx address
- **AR** Auto Receive
- **MA** Monitor All
- **MR hh** Monitor for Rxer hh
- **MT hh** Monitor for Txer hh

Figure 1. ELM322 AT Commands

OBD Commands

If the bytes received on the RS232 bus do not begin with the letters A and T, they are assumed to be commands for the vehicle's OBD bus. The bytes will be tested to ensure that they are valid pairs of hexadecimal digits and, if they are, will be combined into bytes for transmitting to the vehicle. Recall that no checks are made as to the validity of the OBD command – data is simply retransmitted as received.

OBD commands are actually sent to the vehicle embedded within a data message. The J1979 standard requires that every message begin with three header bytes followed by the data bytes, and finally be terminated with a checksum byte, but the ELM322 takes care of this formatting for you. It powers on expecting to be used for OBDII mandated emissions diagnostics, so knows the values necessary for the header bytes, sets them accordingly, and simply has to insert the user's data. If you wish to experiment with some of the more advanced functions, the values used for the header bytes may be changed with AT commands, as discussed previously. To view these extra bytes as they are received, you must turn the header display on by issuing an ATH1 command.

The command portion of most OBD messages is usually only one or two bytes in length, but can occasionally be longer, as the standard allows for as many as seven. The current version of the ELM322 will accept the maximum seven command bytes (or 14 hexadecimal digits) per message, while users of previous versions (v1.x) were limited to only three command bytes. In either case, attempts to send more than the maximum number of bytes allowed will result in a syntax error, with the entire command being ignored and a single question mark printed.

The use of hexadecimal digits for all of the data exchange was chosen as it is the most common data format used in the relevant SAE standards. It is consistent with mode request listings and is the most frequently used format for displaying results. With a little practice, it should not be very difficult to deal in hex numbers, but some may initially find the table in Figure 2 or a calculator to be invaluable. All users will eventually be required to manipulate the results in some way, though (combine bytes and divide by 4 to obtain rpm, divide by 2 to obtain degrees of advance, etc.), and may find a software front-end 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 ELM322 on the RS232 bus. The ELM322 would store the characters as they are received, and when the third character (the carriage return) is received, 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 (with a decimal value of 166). Three header bytes and a checksum byte would be added, so a total of five bytes would be sent to the vehicle. Note that the carriage return character is only a signal to the ELM322, and is not sent on to the vehicle.

After sending a command, the ELM322 listens to the OBD bus for any responses that are directed to it. Each received byte is converted to the equivalent hexadecimal pair of ASCII characters and transmitted on the RS232 port for the user. Rather than send control characters which are unprintable on most terminals, the digits are sent as numbers and letters (e.g. the hex digit 'A' is transmitted as decimal value 65, and not 10).

If there was no response from the vehicle, due to no data being available, or because the command is not supported, a 'NO DATA' message will be sent. See the Error Messages section for a description of this message and others.

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 2. Hex to Decimal Conversion

Talking to the Vehicle

The ELM322 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, you only need to use a terminal program to send bytes to, and receive them from, the vehicle.

SAE standards specify that command 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 standard documents J1979 and J2190, and may also be expanded on by the vehicle manufacturers.

Normally, one is only concerned with the nine diagnostic test modes described in J1979 (although there is provision for more). Note that it is not a requirement for all of them to be supported. 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 : test results, continuously monitored
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 is required to be supported by all vehicles, and can be accessed as follows…

Ensure that the ELM322 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 on, but do not start the vehicle. At the prompt, issue the mode 01 PID 00 command:

```
>01 00
```

A typical response could be as follows:

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

The 41 00 signifies a response (4) from a mode 1 request with 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 which of PIDs 1 through 32 are supported by this mode (1=supported, 0=not). Although this information is not very useful for the casual user, it does serve to show that you are communicating with the vehicle.

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

```
>01 05
```

The response will be of the form:

```
41 05 7B
```

This shows a mode 1 response (41) from PID 05, with value 7B. Converting the hexadecimal 7B to decimal, one gets 7 x 16 + 11 = 123. This represents the current temperature in degrees Celsius, with the zero value offset by 40 degrees to allow operation at subzero temperatures. To convert to the actual coolant temperature, simply subtract 40 from the value. In this case, then, the ECT is 123 - 40 = 83 degrees Celsius.

A final example shows a request for the OBD requirements to which this vehicle was designed. This is PID 1C of mode 01, so at the prompt, type:

```
>01 1C
```

A typical response would be:

```
41 1C 01
```

The returned value (01) shows that this vehicle conforms to OBDII (California ARB) standards. The presently defined responses are:

01 : OBDII (California ARB)
02 : OBD (Federal EPA)
03 : OBD and OBDII
04 : OBD I
05 : not intended to meet any OBD requirements
06 : EOBD (Europe)

Some modes may provide multi-line responses (09, if supported, can display the vehicle's serial number). The ELM322 will attempt to display all responses in these cases, but only if it is allowed sufficient time to process each. There may be occasions when the vehicle responds too quickly to allow time for reprocessing, and lines could be lost.

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

Interpreting Trouble Codes

Likely the most common use that the ELM322 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 our request, and the first data byte (81) is the result that we are looking for. Clearly there would not be 81(hex) or 129(decimal) trouble codes if the vehicle is 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') has been turned on by one of this module's codes (if there are more than one), while the other 7 bits provide the actual number of stored codes. To determine the number of stored codes, then, one needs to subtract 128 (or 80 hex) from the number if it is greater than 128, and otherwise 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 MIL or 'Check Engine' lamp on. The remaining bytes in the response provide information on the types of tests supported by that particular module (see 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 (ATH1) 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 there is only one trouble code here. The response has been padded with 00's as is required by the standard, and the extra 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 first digit of a trouble code 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 1131 would be P1131.

Had there been codes stored by more than one module, or more than three codes stored in the same module, the above response would have consisted of multiple lines. To determine which module is reporting each trouble would then require turning the headers on with an ATH1 command.

Resetting Trouble Codes

The ELM322 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' lamp) 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 ELM322, as it occurs whenever a scan tool is used to reset your codes. Understand that the loss of this data could cause your car to run poorly for a short time while the system recalibrates itself.

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. Recall, though, that the ELM322 does not monitor the content of 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 (the ignition on but the engine not running, etc.) 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 inadvertently issue an 04!

Error Messages

When problems occur, the ELM322 will respond with one of the following short messages. Here is a brief description of each…

BUS BUSY

The ELM322 tried to send the mode command or request for about 0.5 seconds without success. Messages are all assigned priorities, which allows one message to take precedence over another. More important things may have been going on, so try re-issuing your request.

BUS ERROR

An attempt was made to send a message, and the data bus voltage did not change as expected. This is most likely because of a circuit problem (a short or open), so check all of your wiring carefully.

<DATA ERROR

There was a problem with the data checksum (CRC byte), indicating a data error in the line pointed to (the ELM322 still shows you what it received). There could have been a circuit problem, or a noise burst which interfered, so try re-sending the request again.

DATA ERROR

The ELM322 expects at least four bytes for every message, and less than that were received. This may have been caused by the key being turned off, or a loose connection, for example, or by receiving a single byte header message when a three byte header was expected. Any monitoring that was in progress will have been aborted. Try turning the display of headers on to see what was actually sent.

NO DATA

There was no response from the vehicle before a timeout occurred. The mode requested may not be supported, so the vehicle ignored you, or the timeout value was set too short, or possibly the ignition key was not turned to the 'on' position. Try issuing an 01 00 command to be sure that the vehicle is ready to receive commands, and if that works, try adjusting the timeout to a longer value with the Set Timeout AT command.

?

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

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. By the same token, you can do a lot of harm if you are careless, so be very careful.

To see how your vehicle uses the OBD bus, you will have to enter one of the ELM322's monitoring modes. The simplest is the "Monitor All" mode, which is entered into by sending the command AT MA from your terminal program. Once received, the IC will continually display any information it sees on the OBD bus, regardless of transmitter or receiver addresses.

Monitoring modes can only be stopped by sending something over the RS232 connection to the ELM322. It is not critical what you send, as any single character will interrupt the IC, returning it to the command mode (waiting for an input). Note that the character you send is discarded and has no effect on any subsequent commands. The IC will always finish a task in progress (printing a line, for example) before returning to wait for input, so always wait for the prompt character ('>') before continuing to issue other commands.

If the headers are not currently displayed, simply typing ATMA shows only the contents of messages, not the transmitter and receiver addresses. To show who is sending to whom, you will need to first turn headers on (AT H1) before beginning to monitor (AT MA). Either way, you may end up with an overwhelming amount of information that you may want to filter, showing only specific messages.

If, for example, you find that the engine controller's address seems to be 10, you may want to restrict the data displayed to only messages from that ECU. To do so, you would monitor only for messages transmitted from address 10, by issuing AT MT 10 from your terminal program. Only messages with 10 in the third byte of the header will be displayed. Similarly, you may wish to only see messages which are being received by address 3B. To monitor for these, send AT MR 3B and only messages with 3B as the second header byte will be shown.

The ELM322 is somewhat limited in its monitoring abilities, in that it does not have an internal buffer to store OBD bus data which appears while a previous message is being sent to the user. If the bus is very active in your application, there is a chance that some messages may be missed due to this inability to buffer in the background. For this reason, you may want to restrict the amount of RS232 data sent by using the MR or MT commands, or using the 'packed data' mode. For most users, this limitation will not be noticeable.

Computer Control – Using Packed Data

If a person is simply asking a vehicle for the current Diagnostic Trouble Codes, speed is normally not an issue, as data is displayed (essentially) as quickly as it can be read. If interfaced to a computer, however, speed may be important.

The packed data mode is a convenient means to effectively triple the ELM322's data transfer rate while maintaining the connection at 9600 baud. Once entered (with AT PD), all OBD messages will be returned as a single length byte followed by the actual data bytes. There are no space characters sent between bytes, no carriage returns or linefeeds – the data is retransmitted exactly as received from the vehicle (except for the change to 9600 baud). While no longer readable on a terminal, computers will understand the information just the same, and will gain speed through both reduced transfer and conversion times. The ELM322 does not function any differently when in this mode – if the headers are to be displayed, they are sent, if in monitoring mode, data is continually sent, etc. The only difference is in the format in which the OBD responses are returned to the controlling computer.

Often there is no response from the vehicle for a particular request. When in the default (formatted data) mode, this is shown with 'NO DATA' being printed, but while in the packed data mode you will only receive a single length byte of value 0 (zero).

While rare, errors may occasionally be detected in the vehicle's data. Normally, a '<DATA ERROR' would be printed for this, but in the Packed Data mode, the checksum (CRC) errors are identified by setting the most significant bit of the length byte. Because of this, one should always check the length byte for a value of 128 or greater before processing the remainder of the message.

Advanced Data Retrieval – Setting the Headers

Prior to v2.0, the ELM322 used a fixed format for the message headers, allowing only for the retrieval of the mandated diagnostic codes, not allowing the user to change them. The IC is now fully programmable, however, allowing the headers to be changed and a great deal more information to be obtained, if your vehicle supports it. Note that only the OBDII diagnostic codes have been mandated, so there is no requirement for all vehicles to support these extra capabilities.

The diagnostic trouble codes that most people are familiar with are described by SAE standard J1979 (ISO15031-5). This is really a specific instance of the many modes allowed by the J2178-4 standard, which provides for information transfer through what is known as 'functional addressing'. For the OBDII mandated diagnostics, requests are actually made to the functional address 6A, with whatever processor is responsible for this function answering the request. Theoretically many different processors can respond to a single functional request, each contributing their insight as to the information requested.

To retrieve some of this extra information, the function being addressed needs to be known. For example, consider that you have studied the J2178 standards and want to request that the processor responsible for Engine Coolant provide the current Fluid Temperature. You determine that Engine Coolant is functional address 48, you know that your address as a scan tool is normally F1, and knowing that the ELM322 does not generate In-Frame Responses (so only supports message types 8 to 15), you choose A8 as the initial priority/type byte.

Combining the above, then, it is desirable to set the three header bytes to A8 48 F1. This is done with the Set Header command, which would be issued at the prompt as follows:

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

The three header bytes assigned in this manner will stay in effect until changed with another AT SH command, a reset, etc. If the default Auto Receive mode has been selected, the receive address will automatically be set to 49 (the second byte plus one). This is consistent with the functional pairs assigned by J2178-4. If you decide that this is not appropriate for your case, you can always set the receive address to what you wish using the AT SR command. For example, if you wanted to obtain a response that is being sent to address 6B instead, you would use AT SR 6B to override the automatic receive mode. Any receive address selected stays in effect until changed by another AT SR, or reinstatement of the automatic mode.

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 typically look like this:

```
>10

10 2E
```

The response to ID 10 is the byte 2E, in this case. 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 (with AT ST FF).

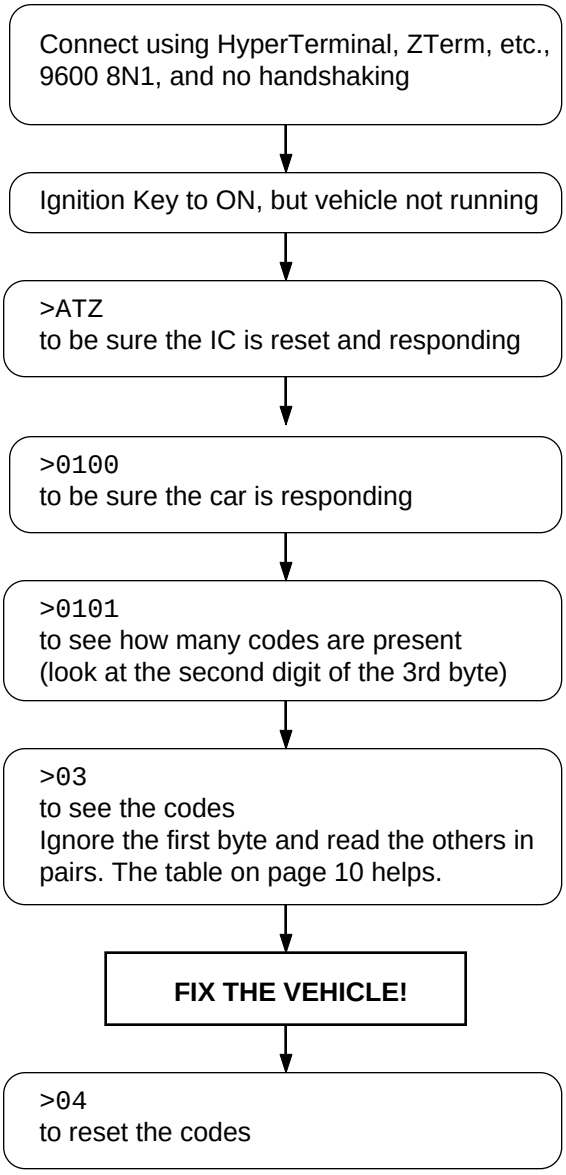
Using the physical addressing modes described by the J2190 standard involves an almost identical process. The main difference is that you must know the physical address of the device that you want to speak to. This address is always the third byte of a message sent by that device, so can be determined by monitoring the headers. Knowing that you wish to talk to address 10, for example, that your physical address is F1, and that for physical node to node addressing EC may be appropriate for the first byte, you would change the header bytes using AT SH EC 10 F1. If Auto Receive is enabled, the receive address will automatically be set to F1, your physical address (the ELM322 knows to do this from the first byte). As before, this header will remain in effect for every message sent until changed to something else.

One caution to note with physical addressing. There are modes which initiate the constant sending of data, and if the ELM322's timeout is set longer than the duration between responses, the ELM322 may return messages forever. In these cases, just like in the monitoring modes, a single character will have to be sent to interrupt the process.

Finally, please note that while we have provided some information on the SAE standards for the examples, Elm Electronics will only reply to requests for clarification on our product's operation, and not on the standards. It is the customer's responsibility to obtain their own information on the relevant standards, and on their vehicle. Requests to Elm Electronics for this information will go unanswered.

Quick Guide for Reading Trouble Codes

If you don't use your ELM322 for some time, this entire data sheet may seem like quite a bit to review if your 'Check Engine' light does eventually come on. The following provides a quick procedure which may prove helpful in that case (note that the '>' is the ELM322's prompt character):



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 3 below. The circuitry described here will be used to connect to this plug without modification to your vehicle.

The male J1962 connector required to mate with a 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 which 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) would definitely not be recommended.

The circuit of Figure 4 on the next page shows how the ELM322 would typically be used. Circuit power is obtained from the vehicle (OBD pins 16 and 5) and, after some minor filtering, is presented to a five volt regulator. Notice that the common point of the regulator is returned to vehicle ground through a diode and an LED, effectively raising the circuit common about 2.5 to 3 volts above that of the vehicle. This gives a net 7.5 to 8 volt positive supply for the OBD bus, as required by the standard (the ground signal shown throughout the schematic refers to the circuit common and not the vehicle's chassis ground).

Note that by offsetting the regulator in this way, the LED and the 750Ω resistor (which provides the current for the LED) become critical components that must not be eliminated. Also, one other subtle result of this is that one must take care not to connect the vehicle's common to the computer's common, as the LED will be shorted out, reducing the supply to 5 volts which is below the required level.

The remaining connection to the OBD bus (pin 2) is the data line required for communications. Data is transmitted onto the bus from the ELM322 via the PNP transistor, the diode, and the 100Ω current limiting resistor (which also provides moderate waveshaping).

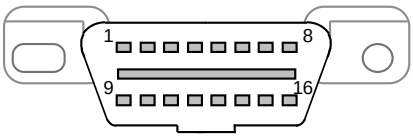


Figure 3. Vehicle Connector

The diode is needed to protect the circuitry from currents which could flow through the transistor if there were high voltages on the bus. Note that the 10KΩ pulldown (loading) resistor returns to vehicle common, providing the data bus with a full (7.5V) voltage swing.

Data is received from the OBD bus and level shifted by the NPN transistor shown connected to pin 4 of the ELM322. Using a transistor this way forces the logic transition point to be at about 3V (the voltage drops of two diodes and an LED) with respect to vehicle common. Had the input been directly connected to pin 4, the threshold would have been approximately 5 volts - much higher than the 3.5 volts specified by the standard.

A very basic RS232 interface is shown connected to pins 5 and 6 of the ELM322. 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 the older 25 pin style, please refer to the web site help pages for the equivalent pins.

RS232 data from the computer is directly connected to pin 5 of the ELM322 through a 47KΩ current limiting resistor. This resistor allows for voltage swings in excess of the supply levels while preventing damage to the ELM322. A single 100KΩ resistor is also shown in this circuit so that pin 5 is not left floating if the computer is disconnected.

Transmission of RS232 data is via the single PNP transistor connected to pin 6. This transistor allows the output voltage to swing between +5V and the negative voltage stored on the 0.1µF capacitor (which is charged to a negative voltage by the computer's TxD line). Although it is a simple circuit, it is quite effective for this type of application.

Finally, the crystal shown connected between pins 2 and 3 is a common television type that can be easily and inexpensively obtained. The 27pF crystal loading capacitors shown are typical values only, and you may have to select other values depending on what is specified for the crystal you obtain.

This circuit is fully functional and complete as shown, and has proven itself to also be quite reliable. Offsetting the two circuit commons may be an unconventional technique (and one of concern to many people at first), but it does work very well in practice.

Many people have written to us saying that their application requires only one common throughout,

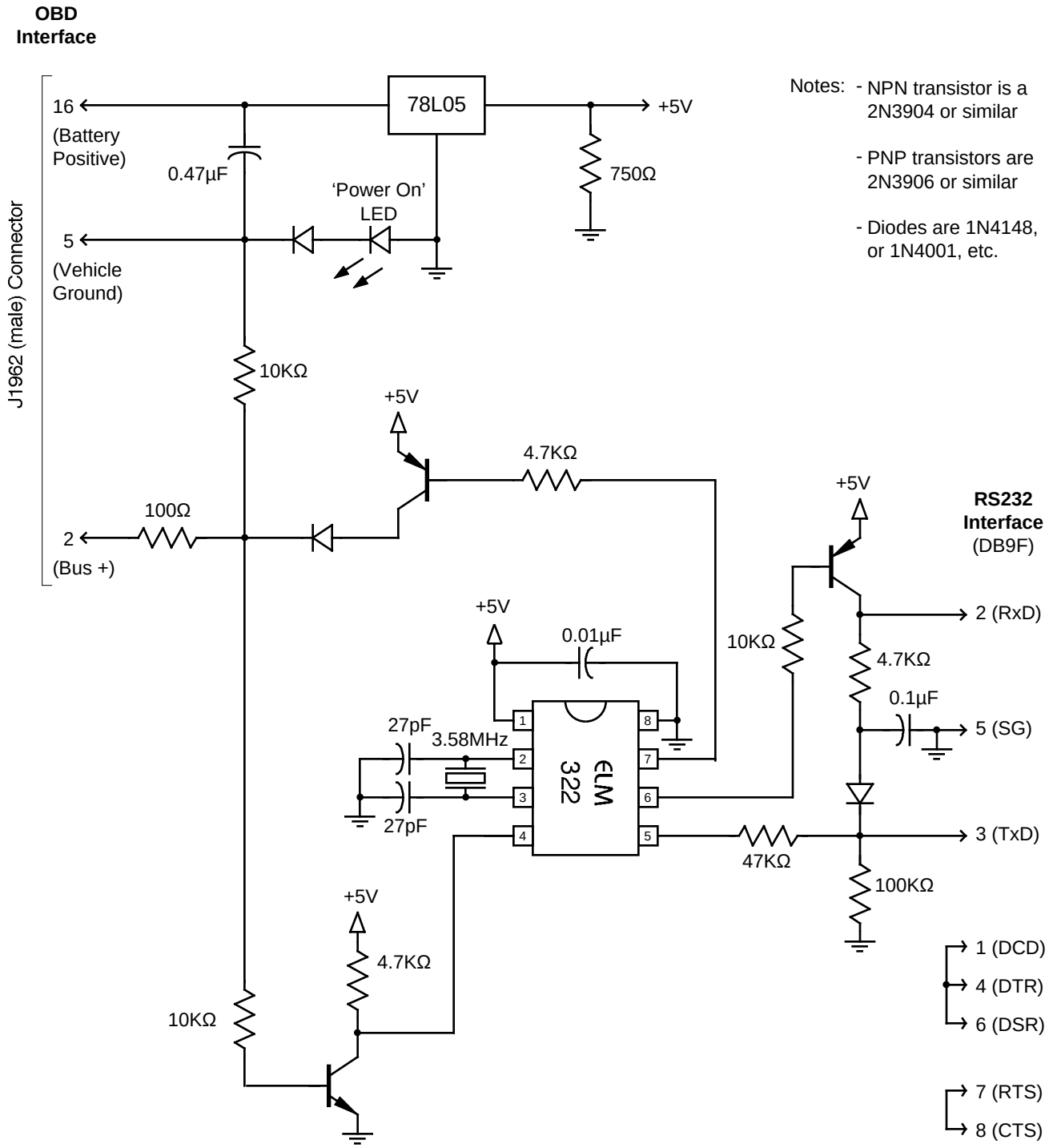


Figure 4. Typical OBD to RS232 Interface

Example Applications (cont'd)

however, and ask for recommendations on how they might accomplish this. Often, they are interfacing to other circuits on a vehicle, or wish to connect directly to a microprocessor, and the different levels that the commons are at are complicating their design. They ask for suggestions as to how they might be able to provide a consistent common throughout.

The circuit of Figure 5 shows one possible way of doing this. It is slightly more complicated than the one of Figure 4, but performs very well.

This circuit uses two voltage regulators in order to create both the 5V needed for the logic, and the 8V needed for the vehicle interface. Actually, the J1850 specification says the output level should be between 6.25V and 8V, so when the transistor and diode drops are taken into account, an 8V supply is ideal. It is recommended that you use an 'L' series regulator for the OBD output supply (ie. a 78L08 as shown), since it will limit the short-circuit current to a low level, should there be problems with the vehicle's wiring.

The OBD output circuitry of Figure 5 looks very similar to that of Figure 4, except that an additional NPN transistor has been added in series between pin 7 and the PNP transistor. This series NPN transistor is used to convert the ELM322's 0V to 5V output swing to a current which then controls the PNP output transistor. This configuration (or something similar) is needed to protect the ELM322 from the 8V, which would damage it.

The OBD input circuitry shown is very similar to that of Figure 4. Note that an NPN is again used as the input device, so that current can only flow into the receive circuitry, not out of it. If a PNP transistor had been chosen, its base current might conceivably be seen as a false active level on some vehicles, which could tie up the OBD bus, and lead to problems.

The only change between this input circuit, and the previous one is the addition of a 2.0KΩ resistor from the base to the emitter of the NPN transistor. This resistor works with the 10KΩ resistor to form a voltage divider, raising the input threshold voltage to about 3V, which is almost identical to that provided by the offset common of Figure 4. A threshold level of about 3V to to 4V is necessary so that vehicle noise does not cause false inputs to the ELM322.

These are the significant differences between the two interface circuits. There are always many ways of accomplishing the same thing, and in this case we have presented two of them for you. They are by no means the only ways of using the ELM322 - you should experiment with your own ideas.

While discussing common questions, another one that is often asked concerns interfacing the ELM322 directly to other logic circuits, without using any RS232 circuitry. Certainly you can, and are encouraged to do so. The ELM322 is simply a 5V CMOS circuit that has plenty of drive capability, and so can be directly connected to most logic families without problems. The one difficulty that some people occasionally encounter is associated with the input level/polarity at pin 5. The ELM322's RS232 Rx input should normally be at V_{SS} (0V) when idle, which is the opposite to what is usually found with serial (UART) interfaces. In order to compensate for the ELM322's inversion, you need to either make provisions for it in your software, or else add an external inverter circuit. If adding and external circuit, it need only be as simple as the NPN transistor and three resistors shown for pin 4 (but you might substitute a 10KΩ resistor for the 2KΩ resistor that is shown). See our web site help pages for more on this.

We hope that this has helped to provide some insight and ideas for your design. While both circuits are fairly simple, they are fully functional and will allow you to do a great deal with an OBDII equipped vehicle. As an experimenter, you may want to expand on these, providing more protection from faults and electrostatic discharge, or providing a different interface for the RS232 connection to your computer or PDA. Then perhaps a Basic program to make it easier to talk to the vehicle, a method to log your findings, a lookup table for trouble codes, and…

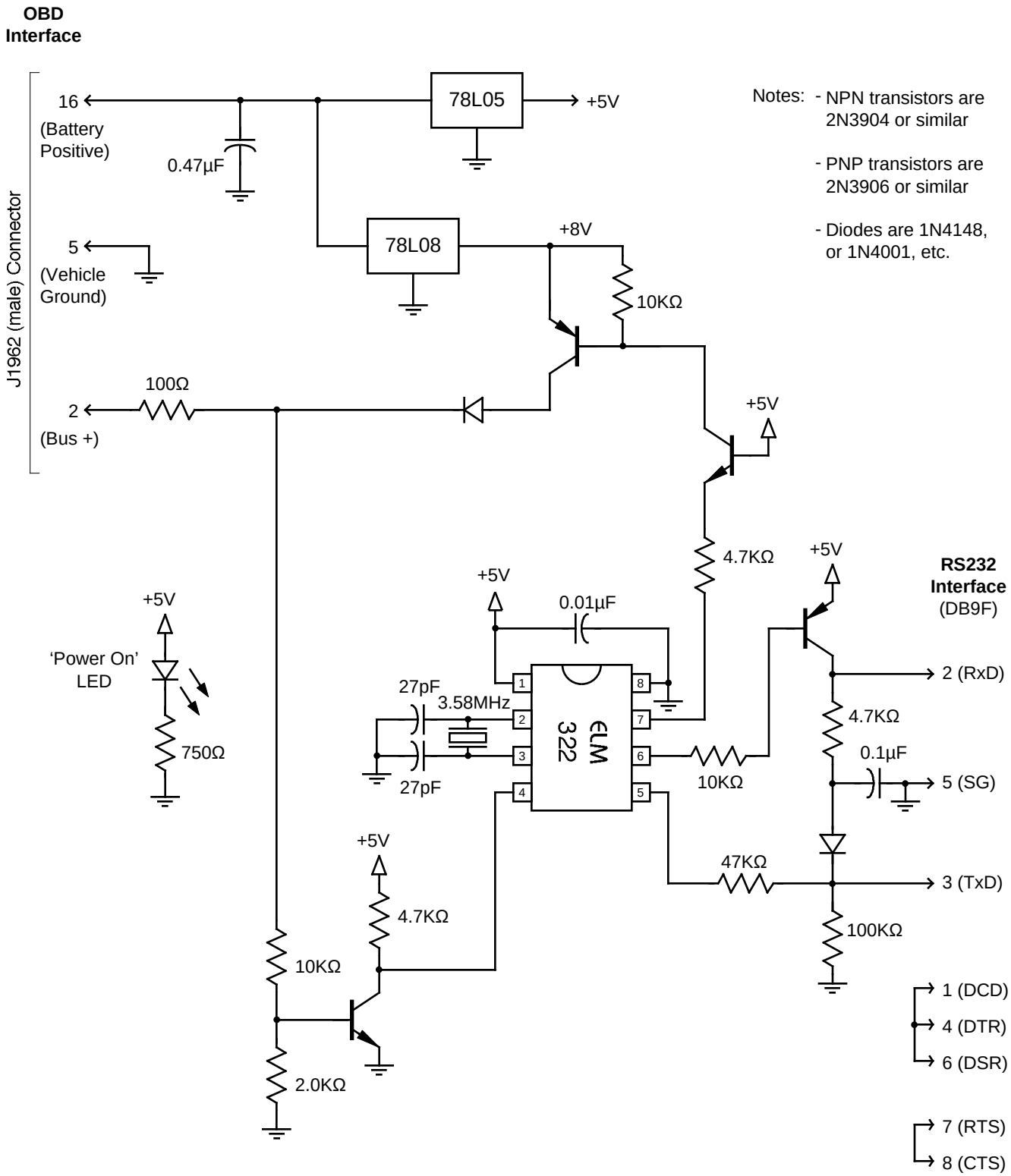


Figure 5. Another Possible Configuration