ELM329
CAN Interpreter

Description

Since 1996, most vehicles have been required to monitor their own emissions performance and to report on it through an On-Board Diagnostics (OBD) port. Initially, several different protocols were used for the transfer of OBD data, but since the 2008 model year (in North America), only one protocol has been allowed - the ISO 15765-4 CAN standard.

The ELM329 is a device that can translate the data from an ISO 15765-4 interface into a form that may be readily used by computers, smart phones, or other devices. In addition, the ELM329 provides support for several other CAN protocols (including the SAE J1939 truck and bus standard), and for sending periodic messages, mixed ID messages, and for monitoring the CAN bus, to name only a few.

The following pages discuss the ELM329's features in detail, how to use it and configure it, as well as providing some background information on the protocols that are supported. There are also schematic diagrams, and circuit construction tips.

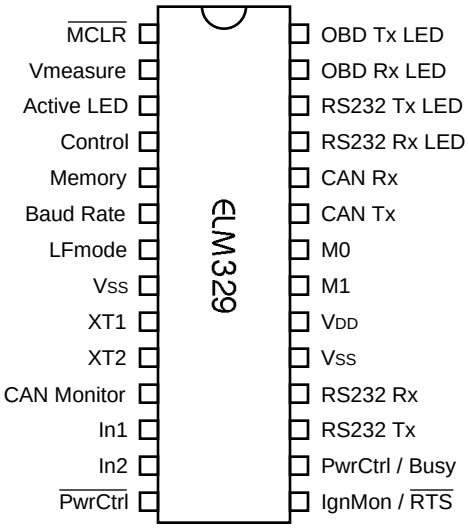
Applications

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

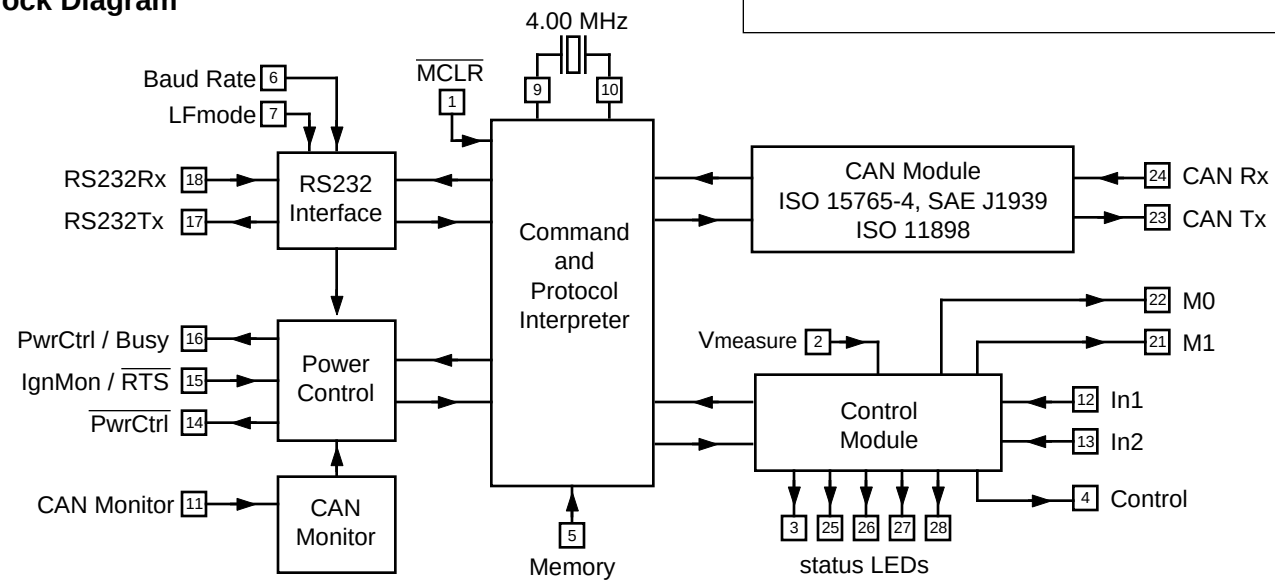
Features

- Power Control with standby mode
- High speed RS232 interface
- Automatically searches for protocols
- Fully configurable with AT commands
- Pin compatible with the ELM327
- Low power CMOS design

Connection Diagram
PDIP and SOIC
(top view)



Block Diagram



Contents

Contents (continued)

Pin Descriptions

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

A momentary (>2µsec) logic low applied to this input will reset the ELM329. 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 V_{DD} signal that is applied to it. The value measured is scaled by a factor of 5.7 (i.e. it assumes an R-4.7R divider) and may be displayed using the AT RV command.

Care must be taken to prevent the voltage from going outside of the supply levels of the ELM329, or damage may occur. If it is not used, this pin should be tied to either V_{DD} or V_{SS}.

Active LED (pin 3)

This output pin is normally at a high level, and is driven to a low level when the ELM329 has determined that it has found a valid (active) protocol. The output is suitable for directly driving most LEDs through a current limiting resistor, or interfacing to other logic circuits. If unused, this pin may be left open-circuited.

Note that the behaviour of this pin when the ELM329 is in the low power mode is controlled by the value of PP 0F, bit 4.

Control (pin 4)

The level at this output may be directly controlled through AT commands. After any reset (powerup, AT Z, etc.), it reverts to low level.

Pin 4 may also be used to provide an output signal that follows the internal CAN monitor output, by setting bit 0 of PP 0F to 1. This is a feature of v2.0 and newer ICs, and was not available with v1.0.

Memory (pin 5)

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

Baud Rate (pin 6)

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

LFmode (pin 7)

This input is used to select the default linefeed mode to be used after a power-up or system reset. If it is at a high level, then by default messages sent by the ELM329 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.

Vss (pin 8)

Circuit common must be connected to this pin.

XT1 (pin 9) and **XT2** (pin 10)

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

When laying out a printed circuit board, you may wish to consider placing a guard ring around the oscillator crystal, pins, and capacitors, to provide a little isolation between them and the other signals (particularly the pin 11 CAN input).

Note that this device has not been configured for operation with an external oscillator, and it expects a crystal to be connected to these pins. Use of an external clock source is not recommended. Also, note that this oscillator is turned off when in the low power or ‘standby’ mode of operation.

CAN Monitor (pin 11)

This input continually watches for a changing (CAN) signal in order to determine if the data bus is active or not. Inactivity for a certain period may be used to cause a switch to the low power (sleep) mode.

Whether there is CAN activity or not may be determined by sending the AT CA command.

In1 and **In2** (pins 12 and 13)

These two inputs may be used for the monitoring of logic level signals. Simple AT commands may be used to read the level at either pin. No special

Pin Descriptions (continued)

amplification is required, as the inputs have Schmitt trigger wave shaping.

$\overline{\text{PwrCtrl}}$ (pin 14)

This output provides a level that is the inverse of that of the PwrCtrl output (pin 16). If the low power mode is disabled (ie if bit 7 of PP 0E is set to '0'), this output still provides the inverse of the level set by PP 0E b6. To provide a 'soft start' feature, pin 14 will always change state 50 msec before pin 16.

IgnMon / **RTS** (pin 15)

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

If both bit 7 and bit 2 of PP 0E are set to '1', this pin will act as an Ignition Monitor. This will result in a switch to the low power mode of operation, if the input goes to a low level, as would happen if the vehicle's ignition were turned off. An internal 'debounce' timer is used to ensure that the ELM329 does not shut down for noise at the input.

When the voltage at pin 15 is again restored to a high level, and a time of 1 or 5 seconds (as set by PP 0E bit 1) passes, the ELM329 will return to normal operation. A low to high transition at pin 15 will always restore normal operation, regardless of the setting of PP 0E bit 2, or whether pin 15 was the initial cause for the low power mode. This feature allows a system to control how and when it switches to low power standby operation, but still have automatic wakeup by the ignition voltage, or by a pushbutton.

If either bit 7 or bit 2 of PP 0E are '0', this pin will function as an active low 'Request To Send' input. This can be used to interrupt the OBD processing in order to send a new command, or if connected to ignition positive, to highlight the fact that the ignition has been turned off. Normally kept at a high level, this input is brought low for attention, and should remain so until the Busy line (pin 16) indicates that the ELM329 is no longer busy, or until a prompt character is received (if pin 16 is being used for power control).

This input has Schmitt trigger wave shaping. By default, pin 15 acts as the RTS interrupt input.

PwrCtrl / **Busy** (pin 16)

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

If bit 7 of PP 0E is a '1' (the default), this pin will function as a Power Control output. The normal state of the pin will be as set by PP 0E bit 6, and the pin will remain in that state until the ELM329 switches to the low power mode of operation. This output is typically used to control enable inputs, but may also be used for relay circuits, etc. with suitable buffering. The discussion on page 84 ('Modifications for Low Power Standby Operation') provides more detail on how to use this output.

If bit 7 of PP 0E is a '0', pin 16 will function as a 'Busy' output, showing when the ELM329 is actively processing a command (the output will be at a high level), or when it is idle, ready to receive commands (the output will be low).

By default, pin 16 provides the PwrCtrl function.

RS232Tx (pin 17)

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

RS232Rx (pin 18)

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

V_{SS} (pin 19)

Circuit common must be connected to this pin.

V_{DD} (pin 20)

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

Pin Descriptions (continued)

M1 (pin 21) and **M0** (pin 22)

These two output pins are provided for use with CAN transceiver ICs, as are typically used for Single Wire CAN applications. The ELM329 will set both outputs to a high level ('Normal') after startup, but the level at these pins may be changed at any time with the AT TM commands, and the level after powerup may be set with PP20.

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

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

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

These four output pins are normally high, and are driven to low levels when the ELM329 is transmitting or receiving data. These outputs are suitable for directly driving most LEDs through current limiting resistors, or interfacing to other logic circuits. If unused, these pins may be left open-circuited.

Note that pin 28 can also be used to turn off all of the Programmable Parameters, if you can not do so by using the normal interface - see page 70 for more details.

Unused Pins

When people only want to implement a portion of what the ELM329 is capable of, they often ask what to do with the unused pins. The rule is that unused outputs may be left open-circuited with nothing connected to them, but unused inputs must always be terminated. The ELM329 is a CMOS integrated circuit that can not have any inputs left floating (or you might damage the IC). Connect unused inputs as follows:

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

The inputs that are shown with an asterisk (*) may be connected to either a High (V_{DD}) or a Low (V_{SS}) level.

Absolute Maximum Ratings

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

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

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

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

Note:

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

Electrical Characteristics

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

Characteristic		Minimum	Typical	Maximum	Units	Conditions
Supply voltage, V_{DD}		4.2	5.0	5.5	V	
V_{DD} rate of rise		0.05			V/ms	see note 2
Average current, I_{DD}	normal		12		mA	ELM329 device only - does not include any load currents
	low power		0.15		mA	
Input logic levels	low	V_{SS}		0.8	V	Pins 5, 6, 7, and 24 only
	high	3.0		V_{DD}	V	
Schmitt trigger input thresholds	rising		2.9	4.0	V	Pins 1, 11, 12, 13, 15 and 18 only
	falling	1.0	1.5		V	
Output low voltage			0.3		V	current (sink) = 10 mA
Output high voltage			4.4		V	current (source) = 10 mA
Brownout reset voltage		2.65	2.79	2.93	V	
A/D conversion time			9		msec	AT RV to beginning of response
Pin 18 low level pulse duration to wake the IC from low power mode		128		-	µsec	
IgnMon debounce time		50	65		msec	
AT LP to PwrCtrl output time			1.0		sec	
LP ALERT to PwrCtrl output time			2.0		sec	
Reset time	AT Z		1040		msec	Measured from the end of the command to the start of the ID message (ELM329 v2.2)
	AT WS		2		msec	

Notes:
1. This integrated circuit is based on Microchip Technology Inc.'s PIC18F2480 device. For more detailed device specifications, refer to the 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 might be obtained through direct connection to solar cells or some charge pump circuits.

Overview

The following describes how to use the ELM329 to obtain information from your vehicle.

We begin by discussing just how to 'talk' to the IC using a PC, then explain how to change options using 'AT' commands, and finally we show how to use the ELM329 to obtain trouble codes (and reset them). For the more advanced experimenters, there are also sections on how to use some of the other features of this product as well.

Using the ELM329 is not as daunting as it first seems. Many users will never need to issue an 'AT' command, adjust timeouts, or change the headers. For most, all that is required is a PC or smart device with a terminal program (such as HyperTerminal or ZTerm), and a little knowledge of OBD commands, which we will provide in the following sections…

Communicating with the ELM329

The ELM329 expects to communicate with a PC through an RS232 serial connection. Although modern computers do not usually provide a serial connection such as this, there are several ways in which a 'virtual serial port' can be created. The most common devices are USB to RS232 adapters, but there are several others such as PC cards, ethernet devices, or Bluetooth to serial adapters.

No matter how you physically connect to the ELM329, you will need a way to send and receive data. The simplest method is to use one of the many 'terminal' programs that are available (HyperTerminal, ZTerm, etc.), to allow typing the characters directly from your keyboard.

To use a terminal program, you will need to adjust several settings. First, ensure that your software is set to use the proper 'COM' port, and that you have chosen the proper data rate - this will be either 9600 baud (if pin 6 = 0V at power up), or 38400 baud (if pin 6 = 5V and PP 0C has not been changed). If you select the wrong 'COM' port, you will not be able to send or receive any data. If you select the wrong data rate, the information that you send and receive will be all garbled, and unreadable by you or the ELM329. Don't forget to also set your connection for 8 data bits, no parity bits, and 1 stop bit, and to set it for the proper 'line end' mode. All of the responses from the ELM329 are terminated with a single carriage return character and, optionally, a linefeed character (depending on your settings).

Properly connected and powered, the ELM329 will energize the five LED outputs in sequence (as a lamp test) and will then send the message:

```
ELM329 v2.2

>
```

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

The '>' character that is shown on the second line is the ELM329's prompt character. It indicates that the device is in the idle state, ready to receive characters on the RS232 port. If you did not see the identification string, you might restart the IC again with the AT Z (reset) command. Simply type the letters A T and Z (spaces are optional), then press the return key:

```
>AT Z
```

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

Characters sent from the computer can either be intended for the ELM329's internal use, or for reformatting and passing on to the vehicle. The ELM329 can quickly determine where the received characters are to be directed by monitoring the contents of the message. Commands that are intended for the ELM329's internal use will begin with the characters 'AT', while OBD commands for the vehicle are only allowed to contain the ASCII codes for hexadecimal digits (0 to 9 and A to F).

Whether it is an 'AT' type internal command or a hex string for the OBD bus, all messages to the ELM329 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 ELM329 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 ELM329

Communicating with the ELM329 (continued)

(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 ELM329 is a protocol interpreter that makes no attempt to assess the OBD messages for validity – it only ensures that hexadecimal digits were received, combined into bytes, then sent out the OBD port, and it does not know if a message sent to the vehicle was in error.

While processing OBD commands, the ELM329 will continually monitor for either an active RTS input (if enabled), or an RS232 character received. Either one can interrupt the IC, quickly returning control to the user, while possibly aborting any initiation, etc. that was in progress. After generating a signal to interrupt the ELM329, software should always wait for either the prompt character ('>' or hex 3E), or a low level on the Busy output before beginning to send the next command.

Finally, it should be noted that the ELM329 is not case-sensitive, so the commands 'ATZ', 'atz', and 'AtZ' are all exactly the same to the ELM329. All commands may be entered as you prefer, as no one method is faster or better. The ELM329 also ignores space characters and all control characters (tab, etc.), so they can be inserted anywhere in the input if that improves readability.

One other feature of the ELM329 is the ability to repeat the last command (AT or OBD) when only a single carriage return character is received. If you have sent a command (for example, 01 0C to obtain the rpm), you do not have to resend the entire command in order to obtain an update from the vehicle - simply send a carriage return character, and the ELM329 will repeat the command for you. The memory buffer only remembers the previous command - there is no provision in the current ELM329 to provide storage for any more.

Please Note:

There is a very small chance that NULL characters (byte value 00) may occasionally be inserted into the RS232 data that is transmitted by the ELM329.

Microchip Technology has reported that some ICs which use the same EUSART as in the ELM329 may, under very specific (and rare) conditions, insert an extra byte (always of value 00) into the transmitted data. If you are using a terminal program to view the data, you should select the 'hide control characters' option if it is available, and if you are writing software for the ELM329, then monitor incoming bytes, and ignore any that are of value 00 (ie. remove NULLs).

AT Commands

Several parameters within the ELM329 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 (ID) bytes. In order to do this, internal 'AT' commands must be used.

Those familiar with PC modems will immediately recognize AT commands as a standard way in which modems are internally configured. The ELM329 uses essentially the same method, always watching the data sent by the PC, looking for messages that begin with the character 'A' followed by the character 'T'. If found, the next characters will be interpreted as an internal configuration or 'AT' command, and will be executed upon receipt of a terminating carriage return character. If the command is just a setting change, the ELM329 will reply with the characters 'OK', to say that it was successfully completed.

Some of the commands require that numbers be provided as arguments, in order to set the internal values. These will always be hexadecimal numbers which must generally be provided in pairs. The hexadecimal conversion chart in the OBD Commands section (page 28) may be helpful if you wish to interpret the values. Also, one should be aware that for the on/off types of commands, the second character is the number 1 or the number 0, the universal terms for on and off.

The remainder of this page, and the next page following provide a summary of all of the commands that the current version of the ELM329 recognizes. A more complete description of each command begins on page 12.

AT Command Summary

ELM329 Options

<CR>	repeat the last command
BRD hh	try Baud Rate Divisor hh
BRT hh	set Baud Rate Timeout
D	set all to Defaults
E0, E1	Echo off, or on*
I	print the version ID
L0, L1	Linefeeds off, or on
LP	go to low power mode
M0, M1	Memory off, or on
RD	Read the stored Data
SD hh	Save Data byte hh
WS	Warm Start (quick software reset)
Z	reset all
@1	display the device description
@2	display the device identifier
@3 ccccccccccc	store the @2 identifier

Note: Settings shown with an asterisk (*) are the default values

Programmable Parameters

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

Voltage Readings

CV dddd	Calibrate the Voltage to dd.dd volts
CV 0000	restore CV value to factory setting
RV	Read the input Voltage

Other

C0, C1	Control Output off*, or on
IGN	read the IgnMon input level
IN1	read INput 1 level
IN2	read INput 2 level

AT Command Summary (continued)

General OBD Commands

AT0, 1, 2	Adaptive Timing off, auto1*, auto2
BD	perform a Buffer Dump
BI	Bypass the Initialization sequence
DP	Describe the current Protocol
DPN	Describe the Protocol by Number
H0, H1	Headers off*, or on
IA	Is this protocol Active?
MA	Monitor All
PC	Protocol Close
R0, R1	Responses off, or on*
S0, S1	printing of Spaces off, or on*
SH xyz	Set Header (11 bit ID) to xyz
SH xxyyzz	Set Header (29 bit ID) to xxyyzz
SH wwxxyyzz	Set Header (29 bit ID) to wwxxyyzz
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
STM1	Set Timer Multiplier to 1*
STM5	Set Timer Multiplier to 5
TA hh	set Tester Address to hh
TP h	Try Protocol h
TP Ah	Try Protocol h with Auto search

J1939 CAN Specific Commands (protocols A to F)

DM1	monitor for DM1 messages
JE	use J1939 Elm data format*
JHF0, JHF1	Header Formatting off, or on*
JS	use J1939 SAE data format
JTM1	set Timer Multiplier to 1*
JTM5	set Timer Multiplier to 5
MP hhhh	Monitor for PGN 0hhhh
MP hhhh n	" " and get n messages
MP hhhhhh	Monitor for PGN hhhhhh
MP hhhhhh n	" " and get n messages

Note: Settings shown with an asterisk (*) are the default values

CAN Specific Commands (protocols 6 to F)

. [1 - 8 bytes]	send bytes with the 11 bit ID
: [1 - 8 bytes]	send bytes with the 29 bit ID
CA	is there CAN Activity at pin 11?
CAF0, CAF1	Automatic Formatting off, or on*
CEA	turn off CAN Extended Addressing
CEA hh	use CAN Extended Address hh
CER hh	set CAN Extended Rx address to hh
CF hhh	set the CAN ID Filter to hhh
CF hhhhhhhh	set the CAN ID Filter to hhhhhhhh
CFC0, CFC1	Flow Controls off, or on*
CM hhh	set the ID Mask to hhh
CM hhhhhhhh	set the ID Mask to hhhhhhhh
CP hh	set CAN Priority to hh (29 bit)
CRA	reset the Receive Address filters
CRA hhh	set CAN Receive Address to hhh
CRA hhhhhhhh	set the Rx Address to hhhhhhhh
CS	CAN Status counts and frequency
CSM0, CSM1	Silent Monitoring off, or on*
D0, D1	display of the DLC off*, or on
FB hh	set the CAN Filler Byte to hh
FC SD [1 - 8 bytes]	Flow Control, Set Data to [...]
FC SM h	Flow Control, Set the Mode to h
FC SH hhh	Flow Control, Set the Header to hhh
FC SH hhhhhhhh	" " to hhhhhhhh
PB xx yy	Protocol B options and baud rate
TM0, 1, 2, 3	set Transceiver Mode to 0,1,2, or 3*
RTR	send an RTR message
V0, V1	use of Variable DLC off*, or on

Periodic Message Commands

SW hh	Set Wakeup interval to hhx20 msec
SW 00	turn off Wakeup messages
W0, W1	Wakeup messages off, or on*
WD [1 - 8 bytes]	set the Wakeup Data bytes
WH hhh	set the Wakeup Header (11 bit)
WH hhhhhhhh	set the Wakeup Header (29 bit)
WT hh	set Wakeup Time to hh x 20 msec
WM0, 1, 2	set the Wakeup Mode to 0*, 1 or 2

AT Command Descriptions

The following describes each AT Command that the current version of the ELM329 supports, in a little more detail. Many of these commands are also described further in other sections:

<CR> [repeat the last command]

Sending a single carriage return character causes the ELM329 to repeat the last command that it performed. This is typically used when you wish to obtain updates to a value at the fastest possible rate - for example, if you send 01 0C to obtain the engine rpm, you need only send a carriage return character each time you wish to receive an update.

Do not use 'AT' before this command.

. [1 - 8 bytes] [send bytes with the 11 bit ID]

If your currently active protocol uses a 29 bit ID, there may be times when you would like to send a single message that has an 11 bit ID. This command is used for that.

A single period ('.') followed by 1 to 8 data bytes will cause the ELM329 to send those data bytes along with the currently defined 11 bit ID. The data will be sent exactly as provided - no formatting bytes or filler bytes will be added, and the number of data bytes sent will be the same as what you provide (so if you need to send 8 bytes as for ISO 15765, then you must provide all 8 of them). The default value used for the 11 bit ID is 7DF, but this may be changed with the AT SH xyz command. See the 'Mixed ID (11 and 29 bit) Sending' section for more information.

A protocol must be active before you can use this command, as the ELM329 needs to know the current baud rate, etc., but it does not have to be a 29 bit one. That is, you may use this 'dot' command with an 11 bit ID protocol, for example, if you needed to send unformatted data along with ISO 15765 formatted data (but the current firmware only allows you to define one 11 bit ID/header).

Do not use 'AT' before this command.

: [1 - 8 bytes] [send bytes with the 29 bit ID]

If your currently active protocol uses an 11 bit ID, there may be times when you would like to send a single message that has a 29 bit ID. This command is used for that.

A single colon (':') followed by 1 to 8 data bytes will cause the ELM329 to send those data bytes along with the currently defined 29 bit ID. The data will be sent exactly as provided - no formatting bytes or filler bytes will be added, and the number of data bytes sent will be the same as what you provide (so if you need to send 8 bytes as for ISO 15765, then you must provide all 8 of them). The default value used for the 29 bit ID is 18 DB 33 F1, but this may be changed with the AT SH xxyyzz or AT SH wwxxyyzz commands. See the 'Mixed ID (11 and 29 bit) Sending' section for more information.

A protocol must be active before you can use this command, as the ELM329 needs to know the current baud rate, etc., but it does not have to be an 11 bit one. That is, you may use this 'colon' command with a 29 bit ID protocol, for example, if you needed to send unformatted data along with ISO 15765 formatted data (but the current firmware only allows you to define one 29 bit ID/header).

Do not use 'AT' before this command.

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

After an OBD request has been sent, the ELM329 waits to see if any responses are coming from the vehicle. The maximum time that it waits is set by the AT ST hh setting, but this setting is purposely a little longer than it needs to be, in order to ensure that the IC will work with a wide variety of vehicles. Although the setting is adjustable, many people do not have the equipment or experience that it would take to determine an optimal value.

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

There are three adaptive timing settings that are available for use. By default, Adaptive Timing option 1 (AT1) is enabled, and is the recommended setting. AT0 is used to disable Adaptive timing (so the timeout is always as set by AT ST), while AT2 is a more aggressive version of AT1 (the effect is more

AT Command Descriptions (continued)

noticeable for very slow connections – you may not see much difference with faster OBD systems). The J1939 protocol does not support Adaptive Timing – it uses fixed timeouts as set in the standard.

BD [perform an OBD Buffer Dump]

All messages sent and received by the ELM329 are stored temporarily in a set of twelve memory storage locations called the OBD Buffer. Occasionally, it may be useful to see the contents of this buffer, perhaps to see why a request failed, to see the header bytes in the last message, or just to learn more of the structure of OBD messages. You can ask at any time for the contents of this buffer to be 'dumped' (ie printed). When you do, the ELM329 sends a length byte (representing the length of the current message in the buffer) followed by the contents of all twelve OBD buffer locations. For example, here's one 'dump':

```
>AT BD
0C 00 00 07 E8 03 41 05 42 00 00 00 00
```

The 0C is the length byte - it tells us that the following 12 bytes are valid. The actual bytes that have been sent or received appear after the length. Note that wakeup (CAN periodic) messages do not use the buffer as an intermediate step, so you are not able to see them with AT BD.

BI [Bypass the Initialization sequence]

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

BRD hh [try Baud Rate Divisor hh]

This command is used to change the RS232 baud rate divisor to the hex value provided by hh, while under computer control. It is not intended for casual experimenting - if you wish to change the baud rate from a terminal program, you should use PP 0C.

Since some interface circuits are not able to be operated at high data rates, the BRD command uses a sequence of sends and receives to test the interface, with any failure resulting in a fallback to the previous baud rate. This allows several baud rates to be tested and a reliable one chosen for the communications. The entire process is described in detail in the 'Using Higher RS232 Baud Rates' section, on pages 47 and 48.

If successful, the actual baud rate (in kbps) will be 4000 divided by the divisor (hh). Note that while setting PP 0C to 00 is valid, sending BRD 00 is not and will result in an error.

BRT hh [set Baud Rate Timeout to hh]

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

C0 and **C1** [Control output off* or on]

These commands are used to set the level at the Control output (pin 4). The AT C0 command sets it to a low logic level (0V), while AT C1 sets it to a high level (5V). After a system reset or wakeup from low power mode (unless PP 0F bit 0 = '1'), the Control output will be reset to a low level. The AT C0 and C1 commands always control the pin 4 output level - it does not matter what PP 0F b0 is set to.

CA [is there CAN Activity at pin 11?]

This command is used to determine if there is a CAN signal present at pin 11 (the CAN Monitor pin). If there is, the response will be the letter 'Y' (for yes), while if there is no signal, the response will be the letter 'N', for no. If there has been no signal detected since the last reset, the output will be a dash ('-').

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

These commands determine whether the ELM329 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 the 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.), without regard to the extra bytes that some CAN systems

AT Command Descriptions (continued)

require. Also, with formatting on, any extra (unused) data bytes that are received in the frame will be removed, and any messages with invalid PCI bytes will be ignored. (When monitoring, however, messages with invalid PCI bytes will be shown, with a '<DATA ERROR' message beside them).

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

You may also see the characters 'FC:' on a line (if you are experimenting). This identifies a Flow Control message that has been sent as part of the multi-line message signalling. Flow Control messages are automatically generated by the ELM329 in response to a 'First Frame' reply, as long as the CFC setting is on (it does not matter if auto formatting is on or not).

Another type of message – the RTR (or 'Remote Transmission 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 one of these frames has been received.

Turning the CAN Automatic Formatting off (CAF0), will cause the ELM329 to print all of the received data bytes. No bytes will be hidden from you, and none will be inserted for you. Similarly, when sending a data request with formatting off, you must provide all of the required data bytes exactly as they are to be sent – the ELM329 will not perform any formatting for you other than to add some trailing 'padding' bytes to ensure that eight data bytes are sent, if required. This allows operation in systems that do not use PCI bytes.

Note that turning the display of headers on (with AT H1) will override some of the CAF1 formatting of the received data frames, so that the received bytes will appear much like in the CAF0 mode (ie. as received). It is only the printing of the received data that will be affected when both CAF1 and H1 modes are enabled, though; when sending data, the PCI byte will still be created for you and padding bytes will still be added. Auto Formatting on (CAF1) is the default setting for the ELM329.

CEA [turn off the CAN Extended Address]

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

CEA hh [set the CAN Extended Address to hh]

Some CAN protocols extend the addressing fields by using the first of the eight data bytes as a target or receiver's address. This type of formatting does not comply with any OBD standard, but by adding it, we allow for some experimentation.

Sending the CEA hh command causes the ELM329 to insert the hh value as the first data byte of all CAN messages that you send. It also adds one more filtering step to received messages, only passing ones that have the Tester Address in the first data byte position (in addition to requiring that ID bits match the patterns set by AT CF and CM, or CRA). The AT CEA hh command can be sent at any time, and changes are effective immediately, allowing for changes of the address 'on-the-fly'. There is a more lengthy discussion of this extended addressing in the 'Using CAN Extended Addresses' section on page 61.

The CEA mode of operation is off by default, and once on, can be turned off at any time by sending AT CEA, with no address. Note that the CEA setting has no effect when J1939 formatting is on.

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

By default, the ELM329 receives responses to CAN extended addressing requests that have the 'tester address' in the first data byte position. The CER command allows you to choose a different receive address.

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 ELM329, otherwise it will be discarded. This version of the CAN Filter command is used 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.

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

This command allows all four bytes (actually 29 bits) of a CAN Filter to be set at once. The 3 most significant bits will always be ignored, and may be given any value. This command may be used to create

AT Command Descriptions (continued)

11 bit ID filters as well, since they are stored in the same locations internally (entering AT CF 00 00 0h hh is exactly the same as entering the shorter AT CF hhh command).

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

The ISO 15765-4 CAN protocol expects a 'Flow Control' message to always be sent in response to a 'First Frame' message, and the ELM329 automatically sends these without any intervention by the user. If experimenting with a non-OBD system, it may be desirable to turn this automatic response off, and the AT CFC0 command has been provided for that purpose. The default setting is CFC1 - Flow Controls on.

Note that during monitoring (ie AT MA), 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 ELM329 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 ELM329 which bits in the filter are relevant, and which ones can be ignored. A 'must match' condition is signalled by setting a mask bit to '1', while a 'don't care' is signalled by setting a bit to '0'. This three digit variation of the CM command is used to provide mask values for 11 bit ID systems (the most significant bit is always ignored).

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

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

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

CP hh [set CAN Priority bits to hh]

This command is used to modify the five most significant bits of a 29 bit CAN ID for sending messages (the other 24 bits are set with one of the AT SH commands). Many systems use these bits to assign a priority value to messages, and to determine the protocol. Any bits provided in excess of the five required are ignored, and not stored by the ELM329 (it only uses the five least significant bits of this byte). The default value for these priority bits is hex 18, which can be restored at any time with the AT D command.

CRA [reset the CAN Rx Addr]

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

CRA xyz [set the CAN Rx Addr to xyz]

Setting the CAN masks and filters can be difficult at times, so if you only want to receive information from one address (ie. one CAN ID), then this command may be very welcome. For example, if you only want to see information from 7E8, simply send AT CRA 7E8, and the ELM329 will make the necessary adjustments to both the mask and the filter for you.

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

CRA wwxxyyzz [set the CAN Rx Addr to wwxxyyzz]

This command is identical to the previous one, except that it is used to set 29 bit CAN IDs, instead of 11. Sending AT CRA will also reverse the changes made by this command.

CS [show the CAN Status counts]

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

AT Command Descriptions (continued)

>FF), you will see 'OFF' rather than a specific count.

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

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

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

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

The ELM329 was designed to be completely silent while monitoring a CAN bus. Because of this, it is able to report exactly what it sees, without colouring the information in any way. Occasionally (when bench testing, or when connecting to a dedicated CAN port), it may be preferred that the ELM329 does not operate silently (ie you may want it to generate ACK bits, etc.), and this is what the CSM command is for. CSM1 turns silent monitoring on (no ACKs), CSM0 turns it off. The default value is CSM1, but it may be changed with PP 21.

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

The voltage reading that the ELM329 shows for an AT RV request can be calibrated with this command. The argument ('dddd') must always be provided as 4 digits, with no decimal point (it assumes that the decimal place is between the second and the third digits).

To use this feature, simply use an accurate meter to read the actual input voltage, then use the CV command to change the internal calibration (scaling) factor. For example, if the ELM329 shows the voltage as 12.2V while you measure 11.99 volts, then send AT CV 1199 and the ELM329 will recalibrate itself for that voltage (it will actually read 12.0V due to digit roundoff). See page 27 ('Reading the Battery Voltage') for some more information on how to read voltages and perform the calibration.

CV 0000 [restore the factory Calibration Value]

If you are experimenting with the CV dddd command but do not have an accurate voltmeter as a reference, you may soon get into trouble. If this happens, you can always send AT CV 0000 to restore the ELM329 to the original calibration value.

D [set all to Defaults]

This command is used to set the options to their default (or factory) settings, as when power is first applied. The last stored protocol will be retrieved from memory, and will become the current setting (possibly closing a protocol that was active). Any settings that the user had made for custom headers, filters, or masks will be returned to their default values, and all timer settings will also be restored to their defaults.

D0 and **D1** [display of DLC off or on]

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

DM1 [monitor for DM1s]

The SAE J1939 Protocol broadcasts trouble codes periodically, by way of the Diagnostic Mode 1 (DM1) messages. This command sets the ELM329 to continually monitor for this type of message for you, even following multi-segment transport protocols if required. Note that a combination of masks and filters could be set to provide a similar output, but they would not allow multiline messages to be detected. The DM1 command adds the extra logic that is needed for multiline messages.

This command is only available when a protocol has been selected for J1939 formatting. It returns an error if attempted under any other conditions.

DP [Describe the current Protocol]

The ELM329 automatically detects a vehicle's OBD protocol, but does not normally report what it is. The DP command is a convenient means of asking what protocol the IC is currently set to (even if it has not yet 'connected' to the vehicle).

If a protocol is chosen and the automatic option is

AT Command Descriptions (continued)

also selected, AT DP will show the word 'AUTO' before the protocol description. Note that the description shows the actual protocol names, and the data rates, it does not provide the numbers used by the protocol setting commands (see DPN for this).

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 (see page 33 for a list of them).

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

These commands control whether or not the characters received on the RS232 port are echoed (retransmitted) back to the host computer. Character echo can be used to confirm that the characters sent to the ELM329 were received correctly. The default is E1 (or echo on).

FB hh [set the CAN Filler Byte to hh]

Most CAN protocols send 8 data bytes in each message. If there are fewer than eight data bytes required to be sent, then 'filler' bytes are added after the data bytes in order to have eight bytes in total. The ELM329 allows you to define the filler byte with PP 26, but that only allows updates when defaults are restored (which can be time-consuming). If you wish to experiment with different protocols, and would like to change the filler byte 'on-the-fly', then use the AT FB command to do so. The FB value reverts to the stored default whenever AT D is issued, or when the ELM329 is reset.

FC SD [1-8 bytes] [Flow Control Set Data to…]

The data bytes that are sent in a CAN Flow Control message may be defined with this command. One to eight data bytes may be specified, with the remainder of the bytes in the message being automatically set to the default CAN filler byte, if more bytes are required by the protocol. Note that no formatting bytes (PCI, etc.) are added by this command - the data is used exactly as provided, except for the filler bytes. AT FC SD is used with Flow Control modes 1 and 2. The default value is 30 00 00.

FC SH xyz [Flow Control Set Header to…]

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

FC SH wwxxyyzz [Flow Control Set Header to…]

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

FC SM h [Flow Control Set Mode to h]

This command sets how the ELM329 responds to First Frame messages when automatic Flow Control responses are enabled. The single digit provided can either be '0' (the default) for fully automatic responses, '1' for completely user defined responses, or '2' for user defined data bytes in the response. Note that FC modes 1 and 2 can only be enabled if you have defined the needed data and possibly ID bytes. If you have not, you will get an error message. More complete details and examples can be found in the Altering Flow Control Messages section (page 59).

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

These commands control whether or not the header (ID and possibly DLC) bytes of information are shown in the responses from the vehicle. These are not normally shown by the ELM329, but may be of interest (especially if you receive multiple responses and wish to determine what modules they were from).

Turning the headers on (with AT H1) actually shows more than just the header bytes – you will see the complete message as transmitted, including the PCI bytes, and the CAN data length code (DLC) if it has been enabled. The current version of this IC does not display the CAN CRC code.

I [Identify yourself]

Issuing this command causes the chip to identify itself, by printing the startup product ID string (currently 'ELM329 v2.2'). Software can use this to determine

AT Command Descriptions (continued)

exactly which integrated circuit it is talking to, without having to reset the IC.

IA [Is this protocol Active?]

This command provides a convenient means of testing whether the current protocol is considered to be active or not. If the protocol is active, the command returns the single character 'Y', while if it is not active, the command returns the character 'N'. Note that a protocol can not be made active by simply monitoring data on the OBD bus. It generally requires that a message be successfully sent and a reply received. See the 'What is an Active Protocol?' section for more information.

IGN [read the IgnMon input level]

This command reads the signal level at pin 15. It assumes that the logic level is related to the ignition voltage, so if the input is at a high level, the response will be 'ON', and a low level will report 'OFF'. This feature is most useful if you wish to perform the power control functions using your own software. If you disable the automatic response to a low input on this pin (by setting bit 2 of PP 0E to 0), then pin 15 will function as the RTS input. A low level on the input will not turn the power off, but it will interrupt any OBD activity that is in progress. All you need to do is detect the 'STOPPED' message that is sent when the ELM329 is interrupted, and then check the level at pin 15 using AT IGN. If it is found to be OFF, you can perform an orderly shutdown yourself.

IN1 [read the level at INput 1]

This command causes the ELM329 to read the logic level at pin 12. If it is at a low level, '0' will be reported, while a high level results in a '1'. The level shown is subject to the hysteresis effects of the Schmitt trigger wave shaping of the input circuitry.

IN2 [read the level at INput 2]

This command causes the ELM329 to read the logic level at pin 13. If it is at a low level, '0' will be reported, while a high level results in a '1'. The level shown is subject to the hysteresis effects of the Schmitt trigger wave shaping of the input circuitry.

JE [enables the J1939 ELM data format]

The J1939 standard requires that PGN requests be sent with the byte order reversed from the standard 'left-to-right' order, which many of us would expect. For example, to send a request for the engine temperature (PGN 00FEEE), the data bytes are actually sent in the reverse order (ie EE FE 00), and the ELM329 would normally expect you to provide the data in that order for passing on to the vehicle.

When experimenting, this constant need for byte reversals can be quite confusing, so we have defined an ELM format that reverses the bytes for you. When the J1939 ELM (JE) format is enabled, and you have a J1939 protocol selected, and you provide three data bytes to the ELM329, it will reverse the order for you before sending them to the ECU. To request the engine temperature PGN, you would send 00 FE EE (and not EE FE 00). The 'JE' type of automatic formatting is enabled by default.

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

When printing responses, the ELM329 normally formats the J1939 ID (ie Header) bits in such a way as to isolate the priority bits and group all the PGN information, while keeping the source address byte separate. If you prefer to see the ID information as four separate bytes (which a lot of the J1939 software seems to do), then simply turn off the formatting with JHF0. The CAF0 command has the same effect (and overrides the JHF setting), but also affects other formatting. The default setting is JHF1.

JS [enables the J1939 SAE data format]

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

Using the above example for engine temperature (PGN 00FEEE) with the data format set to JS, you must send the bytes to the ELM329 as EE FE 00 (this is also known as little-endian byte ordering).

The JS type of data formatting is off by default.

JTM1 [J1939 Timer Multiplier to 1]

This command causes all timeouts set by AT ST to be as set (i.e. to have a multiplier of 1). JTM1 is the default multiplier setting. At one time, the JTM1 command was only used for J1939 protocols, but no longer. It is identical to the STM1 command.

AT Command Descriptions (continued)

JTM5 [J1939 Timer Multiplier to 5]

The JTM5 command multiplies the AT ST setting by a factor of 5, providing settings up to about 5 seconds, rather than 1. At one time, the JTM5 command was only used for J1939 protocols, but no longer. It is identical to the STM5 command, and is off by default.

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

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

Note that while the intention of this command was to tell the ELM329 what to send and what to expect to receive, it was found that many people did not adhere to this. Many would send AT L0, but then provide linefeeds after their command's carriage return. The result would be that the command began executing after the carriage return, but then would be interrupted by the subsequent linefeed character. For this reason, the ELM329 code will always wait after receiving a carriage return character, to see if a linefeed character is coming. This is not noticeable to human operators, but if you want your computer code to run at the fastest speed, it is best to always send a linefeed character immediately after your carriage returns.

LP [go to the Low Power mode]

This command causes the ELM329 to shut off all but 'essential services' in order to reduce the power consumption to a minimum. The ELM329 will respond with an 'OK' (but no carriage return) and then, one second later, will change the state of the PwrCtrl outputs (pins 14 & 16) and will enter the low power (standby) mode. The IC can be brought back to normal operation with an RS232 input, CAN activity, or a rising edge at the IgnMon (pin 15) input, in addition to the usual methods of resetting the IC (power off then on, a low on pin 1, or a brownout). See the 'Low Power Mode' section (page 66) for more information.

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

The ELM329 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 ELM329 to a high logic level).

When the memory function is enabled, each time that the ELM329 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 ELM329 will always start at power up using the same (last saved) protocol.

If the ELM329 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 ELM329 to start in an 'Automatic' protocol search mode, which is the most useful for an unknown environment. ICs come from the factory set to this mode. If, however, you have only one vehicle that you regularly connect to, storing that vehicle's protocol as the default would make the most sense.

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

MA [Monitor All messages]

This command places the ELM329 into a bus monitoring mode, in which it continually monitors for (and displays) all messages that it sees on the OBD bus. It is a quiet monitor, not sending Acknowledge bits or Wakeup (CAN periodic) messages. Monitoring will continue until it is stopped by activity on the RS232 input, or the RTS pin.

To stop the monitoring, simply send any single character to the ELM329, then wait for it to respond with a prompt character ('>'), or a low level output on the Busy pin. (Setting the RTS input to a low level will interrupt the device as well.) Waiting for the prompt is necessary as the response time varies depending on

AT Command Descriptions (continued)

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

All messages that are received by the ELM329 will be printed as found, even if the CAN auto formatting is on. Normally, the automatic formatting will clean up what is displayed, hiding errors, improperly formatted messages, etc. but when monitoring, you will see all messages that pass through the receive filter, and the error messages.

If the filter and/or mask are set (with the CF, CM or CRA commands) before sending AT MA, then the data displayed will be restricted to only those messages that meet the criteria. This is normally desired, but occasionally brings unexpected results when users are not aware. If you truly want to see all data, then you may want to be sure there is no filtering of data (send AT CRA before the AT MA).

The MA monitoring command operates by closing the current protocol (an AT PC is executed internally), then configuring the IC for silent monitoring of the data (no wakeup messages, or acknowledges are sent by the ELM329). When the next OBD command is to be transmitted, the protocol will again be initialized, and you may see messages stating this. 'SEARCHING...' may also be seen, depending on what changes were made while monitoring.

MP hhhh [Monitor for PGN hhhh]

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

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

The AT MP command is only available when a protocol has been selected for SAE J1939 formatting. It returns an error if attempted under any other conditions. Note also that this version of the ELM329 only displays responses that match the criteria, not the requests that are asking for the PGN information.

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

This is very similar to the previous command, but adds the ability to set the number of messages that should be fetched before the ELM329 automatically stops monitoring and prints a prompt character. The value 'n' may be any single hex digit.

MP hhhhhh [Monitor for PGN hhhhhh]

This command is very similar to the MP hhhh command, but it extends the number of bytes provided by one, so that there is complete control over the PGN definition (it does not make the assumption that the Data Page bit is 0, as the MP hhhh command does). This allows for future expansion, should additional PGNs be defined with the Data Page bit set. Note that internally, the filter and mask are set using the values provided, but only the Data Page bit is relevant in the mask - the other bits are ignored. If you need more precise matching of the priority and EDP bits, you might consider the AT CM and AT CF commands to set the filter and mask, then use AT MA.

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

This is very similar to the previous command, but it adds the ability to set the number of messages that should be fetched before the ELM329 automatically stops monitoring and prints a prompt character. The value 'n' may be any single hex digit.

PB xx yy [set Protocol B parameters]

This command allows you to change the protocol B (USER1) options and baud rate without having to change the associated Programmable Parameters. This allows for quicker testing, and program control.

To use this feature, simply set xx to the value for PP 2C (the formatting options), and yy to the value for PP 2D (the baud rate divisor), and send this command. The next time that the protocol is initialized it will use these values.

For example, assume that you wish to monitor data on a bus that uses 11 bit CAN IDs, and operates

AT Command Descriptions (continued)

at 33.3 kbps. If you do not want any special formatting, this means a value of 11000000 (C0 hex) for PP 2C, while 33.3 is 500/15 so 15 decimal or 0F hex is needed for PP 2D. You may send these values to the ELM329 in one command:

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

then monitor:

```
>AT MA
```

If you want to try another protocol (for example, 500 kbps J1939) then simply close the current protocol and send another AT PB command:

```
>AT PC
OK

>AT PB 42 01
OK

>AT MA
```

Values passed in this way do not affect those that are stored in the 2C and 2D Programmable Parameters, and are lost if the ELM329 is reset. If you want to make your settings persist over power cycles, then you must store them in the Programmable Parameter memory for one of the five USER protocols (ie protocols B to F).

PC [Protocol Close]

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

PP hh OFF [turn Prog Parameter hh OFF]

This command disables Programmable Parameter number hh. Any value assigned using the PP hh SV command will no longer be used, and the factory default setting will once again be in effect. The actual time when the new value for this parameter becomes effective is determined by its type. Refer to the Programmable Parameter Summary section (page 70) for more information on the types.

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

It is possible to alter some of the Programmable Parameters so that it may become difficult, or even impossible, to communicate with the ELM329. If this occurs, there is a hardware means of resetting all of the Programmable Parameters at once. Connect a jumper from circuit common to pin 28, holding it there while powering up the ELM329 circuit. Hold it in position until you see the RS232 Receive LED begin to flash (which indicates that all of the PPs have been turned off). At this point, remove the jumper to allow the IC to perform a normal startup. Note that a reset of the PPs occurs quite quickly – if you are holding the jumper on for more than a few seconds and do not see the RS232 receive light flashing, remove the jumper and try again, as there may be a problem with your connection.

PP hh ON [turn Prog Parameter hh ON]

This command enables Programmable Parameter number hh. Once enabled, any value assigned using the PP hh SV command will be used instead of the factory default value. (All of the programmable parameter values are set to their default values at the factory, so enabling a programmable parameter before assigning a value to it will not cause problems.) The actual time when the value for this parameter becomes effective is determined by its type. Refer to the Programmable Parameters section (page 70) for more information on the types.

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

PP hh SV yy [Prog Parameter hh, Set Value to yy]

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

PPS [Programmable Parameter Summary]

The complete range of Programmable Parameters are displayed with this command (even those not yet implemented). Each is shown as a PP number followed by a colon and the value that is assigned to it. This is followed by a single digit – either 'N' or 'F' to show that it is ON (enabled), or OFF (disabled),

AT Command Descriptions (continued)

respectively. See the Programmable Parameters section for a more complete discussion.

R0 and **R1** [Responses off or on]

These commands control the ELM329's automatic receive (and display) of the messages returned by the vehicle. If responses have been turned off, the IC will not wait for a reply from the vehicle after sending a request, and will return immediately to wait for the next RS232 command (the ELM329 does not print anything to say that the send was successful, but you will see a message if it was not).

R0 may be useful to send commands blindly when using the IC for a non-OBD network application, or when simulating an ECU in a learning environment. It is not very useful for normal OBD communications, however, as the purpose of making request is to obtain replies.

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

RD [Read the Data in the user memory]

The byte value stored in the non-volatile user memory (with the SD command) is retrieved with this command. There is only one memory location, so no address is required.

RTR [send an RTR message]

This command causes a special 'Remote Frame' CAN message to be sent. This type of message has no data bytes, and has its Remote Transmission Request (RTR) bit set. The headers and filters will remain as previously set (ie the ELM329 does not make any assumptions as to what format a response may have), so adjustments may need to be made to the mask and filter before sending an RTR. This command must be used with an active CAN protocol (one that has been sending and receiving messages), as it can not initiate a protocol search. Note that the CAF1 setting normally eliminates the display of all RTRs, so if you are monitoring messages and want to see the RTRs, you will have to turn off formatting, or else turn the headers on.

The ELM329 treats an RTR just like any other message sent, and will wait for a response from the vehicle (unless AT R0 has been chosen).

RV [Read the input Voltage]

This initiates the reading of the voltage present at pin 2, and the conversion of it to a decimal voltage. By default, it is assumed that the input is connected to the voltage to be measured through a 470KΩ and 100KΩ resistor divider (with the 100KΩ connected from pin 2 to Vss), and that the ELM329 supply is a nominal 5V. This will allow for the measurement of input voltages up to about 28V, with an uncalibrated accuracy of typically about 2%.

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

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

The ELM329 normally reports ECU responses as a series of hex characters that are separated by space characters (to improve readability), but messages can be transferred much more quickly if every third byte (the space) is removed. While this makes the message less readable for humans, it can provide significant improvements for computer processing of the data, and reduce the amount of data in the send buffer. By default, spaces are on (S1), and space characters are inserted in every response.

SD hh [Save Data byte hh]

The ELM329 is able to save one byte of information for you in a special nonvolatile memory location, which is able to retain its contents even if the power is turned off. Simply provide the byte to be stored, then retrieve it later with the read data (AT RD) command. This location is ideal for storing user preferences, unit ids, occurrence counts, or other information.

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

Each message that is sent by the ELM329 is a combination of a header (ID bits) and data bytes. Since the ID bits need to be changed far less often than the data byes, it makes sense to change them only when needed.

The AT SH xyz command accepts a three digit argument, takes only the right-most 11 bits from that, and uses that for the 11 bit ID when sending standard length ID messages.

AT Command Descriptions (continued)

SH xxyyzz [Set the Header to xxyyzz]

This command provides a means to set three bytes of the 29 bit extended ID. The values passed are used to populate the 24 least significant bits (and the remaining 5 bits are set using the AT CP command). Since the CAN Priority bits do not often change, this three byte/six digit command often provides a slightly faster way to change an extended ID. In addition, it provides compatibility with the large ELM327 software base.

The header bytes (ID bits) in a message are normally assigned values for you (and depending on your application, may never require adjusting), but there may be occasions when it is desirable to change them (particularly if experimenting with physical addressing). If experimenting, it is not necessary but may be better to set the headers after a protocol is active. That way, you can be sure of your starting point before changing the default values.

The header bytes are defined with hexadecimal digits. These remain in effect until set again, or until restored to their default values with the D, WS, or Z commands.

If new values for header bytes are set before the vehicle protocol has been determined, and if the search is not set for fully automatic (ie other than protocol 0), these new values will be used for the header bytes of the first request to the vehicle. If that first request should fail to obtain a response, and if the automatic search is enabled, the ELM329 will then continue to search for a protocol using default values for the header bytes. Once a valid protocol is found, the header bytes will revert to the values assigned with the AT SH command.

SH wwxxyyzz [Set the Header to wwxxyyzz]

All 29 bits of an extended ID (header) may be set at once with this command. Only 29 bits are used - the three most significant bits of the first digit are ignored.

SP h [Set Protocol to h]

This command is used to set the ELM329 for operation using the protocol specified by 'h', and to also save it as the new default. Note that the protocol will be saved no matter what the AT M0/M1 setting is.

The ELM329 supports many different protocols, as listed here (but it's a little misleading, as there is only very minimal support for protocols 1 to 5):

0 - Automatic
1 - SAE J1850 PWM (41.6 kbaud)
2 - SAE J1850 VPW (10.4 kbaud)
3 - ISO 9141-2 (5 baud init, 10.4 kbaud)
4 - ISO 14230-4 KWP (5 baud init, 10.4 kbaud)
5 - ISO 14230-4 KWP (fast init, 10.4 kbaud)
6 - ISO 15765-4 CAN (11 bit ID, 500 kbaud)
7 - ISO 15765-4 CAN (29 bit ID, 500 kbaud)
8 - ISO 15765-4 CAN (11 bit ID, 250 kbaud)
9 - ISO 15765-4 CAN (29 bit ID, 250 kbaud)
A - SAE J1939 CAN (29 bit ID, 250* kbaud)
B - USER1 CAN (11* bit ID, 125* kbaud)
C - USER2 CAN (11* bit ID, 50* kbaud)
D - SAE J1939* CAN (29* bit ID, 500* kbaud)
E - USER4 CAN (11* bit ID, 95.2* kbaud)
F - USER5 CAN (11* bit ID, 33.3* kbaud)

* default settings (user adjustable)

The first protocol shown (0) is a convenient way of telling the ELM329 that the vehicle's protocol is not known, and that it should perform a search for you. It causes the ELM329 to try all protocols if necessary, looking for one that can be initiated correctly. When a valid protocol is found, and the memory function is enabled, that protocol will then be remembered, and will become the new default setting. When saved like this, the automatic mode searching will still be enabled, and the next time the ELM329 fails to connect to the saved protocol, it will again search all protocols for another valid one. Note that some vehicles respond to more than one protocol - during a search, you may see more than one type of response.

The AT SP 0 command is a useful way to reset the search logic when attempting to connect to a vehicle. The ELM329 SP 0 command works like the ELM327 SP 0 command - it is the only one that does not cause an immediate write to EEPROM (which is an unnecessary step if the IC is only to begin searching for another protocol immediately after). If you feel for some reason that you must store a '0' for the protocol, you may send AT SP 00, but it is not necessary.

Protocols 1 to 5 in the above list are only included for compatibility with ELM327 software, and have only very limited functionality. If you use them to try to send a request, nothing is sent, and the ELM329 will return reporting 'NO DATA'. Similarly, if trying to

AT Command Descriptions (continued)

monitor with protocols 1 to 5 (using AT MA), they report no data.

SP Ah [Set Protocol to Auto, h]

This variation of the SP command allows you to choose a starting (default) protocol, while still retaining the ability to automatically search for a valid protocol on a failure to connect. For example, if you think that your vehicle is ISO 15765-4, 11 bit ID and 250 kbaud, you may send the AT SP A8 command to tell the ELM329 to try protocol 8 first, then automatically search for another if that fails.

There is one problem with using this command - the message that you provide is sent using the protocol that you specify, without regard to what baud rate the bus is actually operating at. If you are mistaken about the baud rate, you will cause errors on the bus, resulting in a momentary disruption, which is not desirable. The much preferred method with CAN protocols is to use the AT SP 0 command.

ST hh [Set Timeout to hh]

After sending a request, the ELM329 waits a preset time for a response before it can declare that there was 'NO DATA' received from the vehicle. The same timer setting is also used after a response has been received, while waiting to see if more are coming. The AT ST command allows this timer to be adjusted, in increments of 4 msec (or 20 msec if STM5 or JTM5 have been selected).

When Adaptive Timing is enabled, the AT ST time sets the maximum time that is to be allowed, even if the adaptive algorithm determines that the setting should be longer. In most circumstances, it is best to simply leave the AT ST time at the default setting, and let the adaptive timing algorithm determine what to use for the timeout.

The ST timer is set to 19 hex (25 decimal) by default which gives a time of approximately 100 msec (this default value can be adjusted with PP 03). Note that a value of 00 does not result in a time of 0 msec – it will restore the timer to the default setting.

STM1 [Set the Timer Multiplier to 1]

This command causes all timeouts set by AT ST to be as set (i.e. to have a multiplier of 1). STM1 is the default multiplier setting.

STM5 [Set the Timer Multiplier to 5]

This command causes all timeouts set by the AT ST command to be extended (multiplied) by a factor of 5. This means that instead of the AT ST setting being the value x 4 msec, it will be the value x 20 msec. The default multiplier is 1, not 5.

SW hh [Set Wakeup period to hh]

Once a data connection has been established, some protocols require that there be data flow every few seconds, just so that the ECU knows to maintain the communications path open. If the messages do not appear, the ECU will assume that you are finished, and will close the channel. If this happens, the connection will need to be initialized again in order to reestablish communications. The ELM329 is able to automatically generate wakeup (periodic CAN) messages, in order to maintain a connection.

The time interval between these periodic 'wakeup' messages can be adjusted in 20.48 msec increments using the AT SW hh command, where hh is any hexadecimal value from 00 to FF. The maximum possible time delay of just over 5 seconds results when a value of FF (decimal 255) is used. The default setting of 62 provides a nominal delay of 2 seconds between messages. The ELM329 makes no effort to display replies to these messages (if there are any) but you might be able to see them depending on filter settings etc.

Note that SW 00 is a special command that blocks the sending of the periodic messages. The subsequent sending of AT SW with a setting that is not 00 will allow periodic messages to resume once again. This is explained in more detail in the Periodic (Wakeup) Messages section..

TA hh [set the Tester Address to hh]

This command is used to change the current tester (ie. scan tool) address that is used in the headers, periodic messages, filters, etc. The ELM329 normally uses the value that is stored in PP 06 for this, but the TA command allows you to temporarily override that value.

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

Although this command may appear to work 'on the fly', it is not recommended that you try to change

AT Command Descriptions (continued)

this address after a protocol is active, as the results may be unpredictable.

TM0, TM1, TM2, TM3 [set the Transceiver Mode…]

This command is used to control the voltage levels at pins 21 (M1) and 22 (M0), and are typically used with single wire CAN transceivers. The four modes are:

0: sleep (M1=0, M0=0)

1: high speed (M1=0, M0=1)

2: high voltage wakeup (M1=1, M0=0)

3: normal (M1=1, M0=1)

Note that during low power operation, the M0 and M1 pins will maintain the setting that they had prior to going to low power mode. The previous version of this IC (v1.0) automatically set both pins to a low level, which is a 'sleep' output. As the ELM329 'wakes up' from low power operation, the level at M0 and M1 will be set according to PP 20. For more details on how to use these commands, see page 63.

TP h [Try Protocol h]

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

TP Ah [Try Protocol h with Auto]

This command is very similar to the AT TP command above, except that if the protocol that is tried should fail to initialize, the ELM329 will then automatically sequence through the other protocols, attempting to connect to one of them.

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

These commands modify the current CAN protocol settings to allow the sending of variable data length messages (just as setting bit 6 of the CAN Options PP does for protocols B to F). This allows experimenting with variable data length messages for any of the CAN protocols, without having to change the Programmable Parameter. The V1 command will always override any protocol setting, and force a variable data length message. The default setting is V0, providing data lengths as determined by the protocol.

W0 and **W1** [Wakeups off or on]

These commands allow periodic CAN messages (wakeups) to be quickly enabled or disabled. The commands simply control the sending of the messages, and have no affect on internal timers, etc. See the 'Periodic (Wakeup) Messages' section for more information. W1 is selected by default.

WD [1 to 8 bytes] [set Wakeup Data to…]

This command allows you to define the data bytes that will be sent in the wakeup (CAN periodic) message. Any number of bytes from 1 to 8 may be defined, and will be sent exactly as provided. That is, no formatting will be applied, no filler bytes will be added, and the data length code will be set to the number of data bytes provided (even if the protocol expects 8 bytes). If you do need to send 8 bytes, you will need to provide 8 bytes. The default setting for WD is 01 3E 00 00 00 00 00 00.

WH xyz [set Wakeup Header to…]

The AT WH xyz command accepts a three digit argument, takes only the right-most 11 bits from that, and uses that for the 11 bit Wakeup Header. You may define a wakeup message to use an 11 bit ID even if the current protocol uses 29 bit IDs. The last assigned wakeup header (11 or 29 bit) determines whether the wakeup message will use standard 11 bit, or extended 29 bit IDs. The default setting for WH is 7DF.

WH wwxxyyzz [set Wakeup Header to…]

This AT WH command accepts a four byte (eight digit) argument, takes only the right-most 29 bits from it, and uses that for the 29 bit Wakeup Header. You may define a wakeup message to use a 29 bit ID even if the current protocol uses 11 bit IDs. The last assigned wakeup header (11 or 29 bit) determines whether the wakeup message will use standard 11 bit, or extended 29 bit IDs. By default, the wakeup header is initially 11 bit.

WM0, WM1, WM2 [set the Wakeup Mode…]

This command is used to set the Wakeup Mode.

AT Command Descriptions (continued)

The modes are defined as:

0: wakeups are off (disabled)

1: wakeups are sent at a constant rate

2: wakeups are sent if no other message has been sent in the AT SW time.

The default value for this option is 0 (off). If you wish to change this, you will need to change PP 23.

WS [Warm Start]

This command causes the ELM329 to perform a complete reset which is very similar to the AT Z command, but does not include the power on LED test. Users may find this a convenient way to quickly 'start over' without having the extra delay of the AT Z command (it is about 1000x faster than AT Z).

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

WT hh [set the Wakeup Timer to hh]

This command simply adjusts the Wakeup Timer timeout setting. It is similar to the AT SW command, but does not affect the enabling of message sends and does not reset the interval timer. Note that the timer count is compared to this setting, so changing the setting will affect any timing currently in progress.

Z [reset all]

This command causes the chip to perform a complete reset as if power were cycled off and then on again. All settings are returned to their default values, and the chip will be put into the idle state, waiting for characters on the RS232 bus. Any baud rate that was set with the AT BRD command will be lost, and the ELM329 will return to the default baud rate setting.

@1 [display the device description]

This command displays the device description string. The default text is 'CAN Interpreter'.

@2 [display the device identifier]

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

@3 cccccccccccc [store the device identifier]

This command is used to set the device identifier code. Exactly 12 characters must be sent, and once written to memory, they can not be changed (ie you may only use the @3 command one time). The characters sent must be printable (ascii character values 00x21 to 0x5F inclusive).

Reading the Battery Voltage

Before learning the OBD Commands, we will show an example of how to use an AT Command. We will assume that you have built (or purchased) a circuit which is similar to that of Figure 9 in the Example Applications section (page 79). This circuit provides a connection to read the vehicle's battery voltage, which many will find very useful.

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

```
>AT RV
```

Note that we used upper case characters for this request, but it was not required, as the ELM329 will accept upper case (AT RV) as well as lower case (at rv) or any combination of these (At rV). It does not matter if you insert space characters (' ') within the message either, as they are ignored by the ELM329.

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

```
12.6V

>
```

The accuracy of this reading depends on several factors. As shipped from the factory, the ELM329 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 ELM329 shows. Many quality digital multimeters can do this, but you should verify the accuracy before making a change.

Let us assume that you have connected your accurate multimeter, and you find that it reads 12.47V. The ELM329 is a little high at 12.6V, and you would like it to read the same as your meter. Simply calibrate the ELM329 to the measured voltage using the CV command:

```
>AT CV 1247
OK
```

Note that you should not provide a decimal point in the CV value, as the ELM329 knows that it should be between the second and the third digits.

At this point, the internal calibration values have been changed (ie. written to EEPROM), and the ELM329 now knows that the voltage at the input is actually 12.47V. To verify that the changes have taken place, simply read the voltage again:

```
>AT RV
12.5V
```

The ELM329 always rounds off the measurement to one decimal place, so the 12.47V actually appears as 12.5V (but the second decimal place is maintained internally for accuracy and is used in the calculations).

The ELM329 may be calibrated with any reference voltage that you have available, but note that the CV command always expects to receive four characters representing the voltage at the input. If you had used a 9V battery for your reference, and it is actually 9.32V, then you must add a leading zero to the actual voltage when calibrating the IC:

```
>AT CV 0932
OK
```

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

```
>AT CV 0000
OK
```

The other AT Commands are used in the same manner. Simply type the letters A and T, then follow with the command you want to send and any arguments that are required. Then press return (or enter, depending on your keyboard). Remember - you can always insert space characters as often as you wish if it improves the readability for you, as they are ignored by the ELM329.

OBD Commands

If the bytes that you send to the ELM329 do not begin with the letters ‘A’ and ‘T’, they are assumed to be OBD commands for the vehicle. Each pair of ASCII bytes will be tested to ensure that they are valid hexadecimal digits, and will then be combined into data bytes for transmitting to the vehicle.

Commands to the vehicle are actually sent embedded in a data packet. The packet consists of header bytes (ie CAN ID bits), as well as checksum and other bits as defined by the ISO standards. The ELM329 adds these extra bits and bytes to your message as required - you do not normally have to even consider them. If you do want to change the ID bits or data lengths at some point, there is a mechanism to do so (see the ‘Setting the Header / ID Bits’ section).

Most OBD commands are only one or two bytes in length, but some can be longer. The current version of the ELM329 will accept up to eight data bytes to be sent (there is no way to send any more bytes with this version). Attempts to send more than eight bytes will result in an error – the entire command is then ignored and a single question mark printed.

Hexadecimal digits are used for all of the data exchange with the ELM329 because it is the data format used most often in the OBD standards. Most mode request listings use hexadecimal notation, and it is the format most frequently used when results are shown. With a little practice, it should not be very difficult to deal in hex numbers, but some people may want to use a table such as Figure 1, or keep a calculator nearby. Dealing with the hex digits can not be avoided - eventually all users need to manipulate the results in some way (combining bytes and dividing by 4 to obtain rpm, dividing by 2 to obtain degrees of advance, converting temperatures, etc.).

As an example of sending a command to the vehicle, assume that A6 (or decimal 166) is the command that is required to be sent. In this case, the user would type the letter A, then the number 6, then would press the return key. These three characters would be sent to the ELM329 by way of the RS232 port. The ELM329 would store the characters as they are received, and when the third character (the carriage return) was received, would begin to assess the other two. It would see that they are both valid hex digits, and would convert them to a one byte value (the decimal value is 166). The header/ID bytes would then be added, and the complete message would then be sent to the vehicle. Note that the carriage return character is only a signal to the ELM329, and is never sent to the vehicle.

After sending the command, the ELM329 listens on the OBD bus for replies, looking for ones that are directed to it. If a message address matches, the received bytes will be converted to ascii characters and sent on the RS232 port to the user, while messages received that do not have matching addresses will be ignored.

The ELM329 will continue to wait for messages addressed to it until there are none found in the time that was set by the AT ST command. As long as messages continue to be received, the ELM329 will continue to reset this timer, and look for more. Note that the IC will always respond to a request with some reply, even if it is to say ‘NO DATA’ (meaning that there were no messages found, or that some were found but they did not match the receive criteria).

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

Figure 1. Hex to Decimal Conversion

Talking to the Vehicle

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

The SAE J1979 standard currently defines ten possible diagnostic test modes, which are:

01 - show current data
02 - show freeze frame data
03 - show diagnostic trouble codes
04 - clear trouble codes and stored values
05 - test results, oxygen sensors
06 - test results, non-continuously monitored
07 - show 'pending' trouble codes
08 - special control mode
09 - request vehicle information
0A - request permanent trouble codes

Vehicles are not required to support all of the modes, and within modes, they are not required to support all possible PIDs (some of the first OBDII vehicles only supported a very small number of them). Within each mode, PID 00 is reserved to show which PIDs are supported by that mode. Mode 01, PID 00 must be supported by all vehicles, and can be accessed as follows…

Ensure that your ELM329 interface is properly connected to the vehicle, and powered. Most vehicles will not respond without the ignition key in the ON position, so turn the ignition to on, but do not start the engine. If you have been experimenting, the state of your interface may be unknown, and you may wish to reset it by sending:

```
>AT Z
```

If you have just powered up the ELM329 circuit, you do not need to do this (as it happens automatically with every power on). You will see the interface LEDs flash, and then the IC should respond with 'ELM329 v2.2', followed by a prompt character. At this point, you may choose a protocol that the ELM329 should connect with, but it is usually best to simply select protocol '0' which tells the IC to search for one:

```
>AT SP 0
```

That's all that you need to do to prepare the ELM329 for communicating with a vehicle, and often you do not even need to do that - most times you can simply jump ahead to the next step. You do not need to know anything about the protocol that your vehicle uses - the ELM329 will determine that for you.

At the prompt, issue the mode 01 PID 00 command:

```
>01 00
```

The ELM329 should say that it is 'SEARCHING...' for a protocol, then it should print a series of numbers, similar to these:

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

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

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

```
>01 05
```

The response will be of the form:

```
41 05 7B
```

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

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

```
>01 0C
```

Talking to the Vehicle (continued)

If the engine is running, the response might be:

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

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

Note that these examples asked the vehicle for information without regard for the type of OBD protocol that the vehicle uses. This is because the ELM329 takes care of all of the data formatting and translation for you. Unless you are going to do more advanced functions, there is really no need to know what the protocol is.

The above examples showed only a single line of response for each request, but the replies often consist of several separate messages, either from multiple ECUs responding, or from one ECU providing messages that need to be combined to form one response (see 'Multiline Responses' on page 39). In order to be adaptable to this variable number of responses, the ELM329 normally waits to see if any more are coming. If no response arrives within a certain time, it assumes that the ECU is finished. This same timer is also used when waiting for the first response, and if that never arrives, causes 'NO DATA' to be printed.

If you know how many responses to expect from a request, it is possible to speed up the retrieval of information a little. That is, if the ELM329 knows how many lines of data to receive, it knows when it is finished, so does not have to go through the final timeout, waiting for data that is not coming. Simply add a single hex digit after the OBD request bytes - the value of the digit providing the maximum number of responses to obtain, and the ELM329 does the rest. For example, if you know that there is only one response coming for the engine temperature request that was previously discussed, you can now send:

```
>01 05 1
```

and the ELM329 will return immediately after obtaining only one response. This may save a considerable amount of time, as the default time for the AT ST timer is 100 msec. (The ELM329 still sets the timer after sending the request, but that is only in case the single response does not arrive.)

For protocols other than J1939, make sure that you know how many lines of data to expect when using this method, not how many responses (J1939 is able to use the count for the number of messages). For example, consider a request for the vehicle identification number (VIN). This number is 17 digits long, and typically takes 5 lines of data to be represented. It is obtained with mode 09, PID 02, and should be requested with:

```
>09 02
```

or with:

```
>09 02 5
```

if you know that there are five lines of data coming. If you should mistakenly send 09 02 1, you will only receive the first few bytes of the VIN.

This ability to specify the number of responses was really added with the programmer in mind. An interface routine can determine how many responses to expect for a specific request, and then store that information for use with subsequent requests. That number can then be added to the requests and the response time can be optimized. For an individual trying to obtain a few trouble codes, the savings are not really worth the trouble, and it's easiest to use the old way to make a request (ie do not put the single digit after the request).

Hopefully this has shown how typical requests are made using the ELM329. If you are looking for more information on modes and PIDs, it is available from the SAE (www.sae.org), from ISO (www.iso.org), or from various other sources on the web.

Interpreting Trouble Codes

Likely the most common use that the ELM329 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 for this 'ECU'. In order to calculate the number of stored codes when the MIL is on, simply subtract 128 (or 80 hex) from the number. At one time, cars had only one main engine controller so ECU was short for 'Engine Control Unit'. Modern vehicles now often have several control units, and ECU usually refers to an 'Electronic Control Unit'.

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 tests that are supported by that ECU (see the SAE J1979 document for further information).

In this instance, there was only one line to the response, but if there were codes stored in other modules, they would each provide a line of response. To determine which module is reporting, you need to turn the 'headers' on (with AT H1) which then shows the ID bits associated with the message.

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

```
>03
```

A response to this could be:

```
43 01 03 02
```

The '43' in the above response simply indicates that this is a response to a mode 03 request. The next byte (the '01') says that 1 trouble code follows, while the remaining two bytes provide the actual trouble code (0302).

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 (0302), the first digit (0) would then be replaced with P0, and the 0302 reported would become P0302 (which is the code for an 'cylinder #2 misfire detected').

If there had been no trouble codes in the above example, the ECU would have told you so. The response would typically look like:

```
>03
43 00
```

That's about all there is to reading trouble codes. With a little practice, you will find it to be quite straight-forward.

Resetting Trouble Codes

The ELM329 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 (among other things):

- 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
- clear the status of the system monitoring tests
- delete on-board test results
- but will not erase permanent (mode 0A) trouble codes (these are reset by the ECU only)

Clearing of all of this data is not unique to the ELM329 – it occurs whenever any scan tool is used to reset the codes. The biggest problem with losing this data is that your vehicle may run poorly for a short time, while it performs a recalibration.

To avoid inadvertently erasing stored information, the SAE specifies that scan tools must verify that a mode 04 is intended ('Are you sure?') before actually sending it to the vehicle, as all trouble code information is immediately lost when the mode is sent. Remember that the ELM329 does not monitor the content of the messages, so it will not know to ask for confirmation of the mode request – this would have to be the duty of a software interface, if one is written.

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

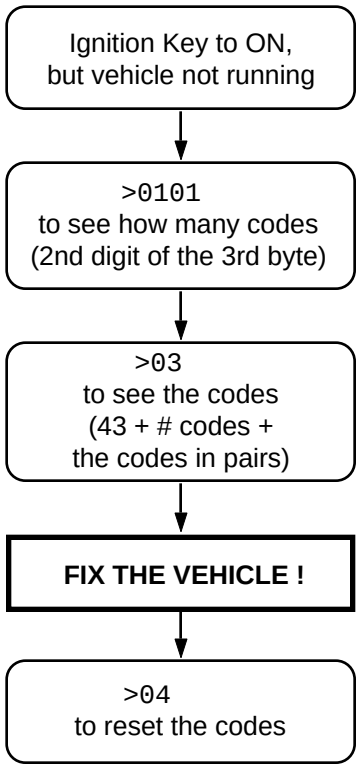
That is all there is to clearing trouble codes. Once again, do not accidentally send the 04 code!

Quick Guide for Reading Trouble Codes

If you do not use your ELM329 for some time, this entire data sheet may seem like quite a bit to review when your 'Check Engine' light eventually comes on, and you just want to know why. We offer this section as a quick guide to the basics that you will need.

To get started, connect the ELM329 circuit to your PC or PDA and communicate with it using a terminal program such as HyperTerminal, ZTerm, ptelnet, or a similar program. It should normally be set to either 9600 or 38400 baud, with 8 data bits, and no parity or handshaking.

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



Selecting Protocols

The ELM329 supports several different OBD protocols (see Figure 2, at right). This is a little misleading however, as the ELM329 only provides very minimal support for protocols 1 to 5 - they are only included so that most ELM327 software will still work with the ELM329.

The ELM329 really only provides support for CAN protocols 6 to F. You may never need to actually select one of these, since the factory settings cause an automatic search to be performed for you, and the protocol is activated if it seems appropriate. If experimenting, you may wish to be able to select a protocol, however.

For example, if you know that your vehicle uses the CAN (ISO 15765-4) standard, with an 11 bit ID and a rate of 500kbps (i.e. protocol 6), then you may want the ELM329 to use only that one, and no others. If that is what you want, simply use the 'Set Protocol' AT Command as follows:

```
>AT SP 6
OK
```

From that point on, the default protocol (used after every power-up or AT D command) will be protocol 6 (or whichever one that you have chosen). Verify this by asking the ELM329 to describe the protocol:

```
>AT DP
ISO 15765-4 (CAN 11/500)
```

Now what happens if your friend has a vehicle that uses a different baud rate? How do you now use the ELM329 interface for that vehicle, if it is set for your car?

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

```
>AT SP A6
OK

>AT DP
AUTO, ISO 15765-4 (CAN 11/500)
```

Now, the ELM329 will try protocol 6, but will then automatically begin searching for another protocol should the attempt to connect with protocol 6 fail (as might happen when you try to connect to your friend's vehicle).

The Set Protocol commands cause an immediate write to the internal EEPROM, before even attempting to connect to the vehicle. This write is time-consuming,

Protocol	Description	
0	Automatic	
1	SAE J1850 PWM (41.6 kbaud)	not functional
2	SAE J1850 VPW (10.4 kbaud)	
3	ISO 9141-2 (5 baud init)	
4	ISO 14230-4 KWP (5 baud init)	
5	ISO 14230-4 KWP (fast init)	
6	ISO 15765-4 CAN (11 bit ID, 500 kbaud)	
7	ISO 15765-4 CAN (29 bit ID, 500 kbaud)	
8	ISO 15765-4 CAN (11 bit ID, 250 kbaud)	
9	ISO 15765-4 CAN (29 bit ID, 250 kbaud)	
A	SAE J1939 CAN (29 bit ID, 250* kbaud)	
B	User1 CAN (11* bit ID, 125* kbaud)	
C	User2 CAN (11* bit ID, 50* kbaud)	
D	SAE J1939* CAN (29* bit ID, 500* kbaud)	
E	User4 CAN (11* bit ID, 95.2* kbaud)	
F	User5 CAN (11* bit ID, 33.3* kbaud)	

*user adjustable

Figure 2. ELM329 Protocol Numbers

affects the setting for the next powerup, and may not actually be appropriate, if the protocol selected is not correct for the vehicle. To allow a test before a write occurs, the ELM329 offers one other command - the Try Protocol (TP) command.

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

```
>AT TP A6
OK
```

Many times, it is very difficult to even guess at a protocol to try first. In these cases, it is best to simply let the ELM329 decide what to use. This is done by telling it to use protocol 0 (with either the SP or the TP commands).

To have the ELM329 automatically search for a

Selecting Protocols (continued)

protocol to use, simply send:

```
>AT SP 0
OK
```

and when the next OBD command is to be sent, the ELM329 will automatically look for one that responds. You will see a 'SEARCHING...' message, followed by a response, after which you can ask the ELM329 what protocol it found (by sending AT DP).

The ELM329 always searches in the order set by the protocol numbers (ie 6, 7, 8, etc.). Note that the IC only appears to provide some support for protocols 1 to 5, but it never actually sends messages using them - all searches start with protocol 6.

The automatic search works well with OBDII systems, but may not be what you need if you are experimenting. During an automatic search, the ELM329 ignores any headers that you have previously defined (since there is always a chance that your headers may not result in a response), and it uses the default OBD header values for each protocol. It will also use standard requests (ie 01 00) during the searches. If this is not what you want, the results may be a little frustrating.

To use your own header (and data) values when attempting to connect to an ECU, do not tell the ELM329 to use protocol 0. Instead, tell it to either use only your target protocol (ie. AT SP n), or else tell it to use yours with automatic searches allowed on failure (ie AT SP An). Then send your request, with headers assigned as required. The ELM329 will then attempt to connect using your headers and your data, and only if that fails (and you have chosen the protocol with AT SP An) will it search using the standard OBD default values.

In general, 99% of all users find that enabling the memory (setting pin 5 to 5V) and using the 'Auto' option when searching (you may need to send AT SP 0) works very well. After the initial search, the protocol used by your vehicle becomes the new default, but it is still able to search for another, without your having to say AT SP 0 again.

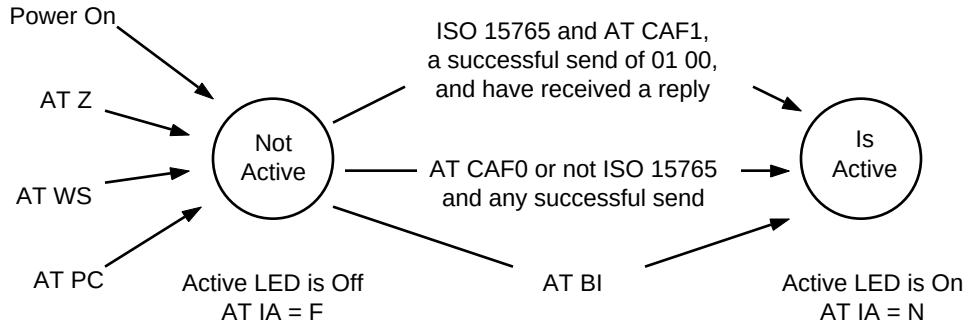
What is an Active Protocol?

The ELM329 needs to know if a protocol is 'active' or not in order to control its' behaviour. An active protocol is one that has been proven to be valid, as far as baud rate, data format, etc. Generally, this is done through the automatic search mechanism, but it may also be determined or set in other ways.

Until a protocol is considered active, there are certain things that may not be done. For example, you may not send RTRs or single '.' or ':' CAN messages. The 'active' state is also used by the monitoring commands in order to know if they should scan for other protocols or stay with the current one. The AT D command uses it in much the same way - if the current protocol is active, it either stays with it, or resets to the saved default protocol.

The Active LED output, and the AT IA command response follows the state of the internal 'active' flag. At any point in time, you may then visually see if the protocol is active, or query the IC as to whether it is.

Just what makes a protocol active then? See the chart below for a diagram of the ELM329's internal logic for it.



OBD Message Formats

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

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

A concern when sending any message is that errors might occur in the transmission, and the received data may be falsely interpreted. To detect errors, all of the protocols provide some form of check on the received data. This may be as simple as a sum calculation (ie a 'running total' of byte values) that is sent at the end of a message. If the receiver also calculates a sum as bytes are received, then the two values can be compared and if they do not agree, the receiver will know that an error has occurred. CAN systems use a special kind of checksum called a Cyclic Redundancy Check (or 'CRC').

The OBD data bytes are thus normally encapsulated within a message, with 'header' bytes at the beginning, and a 'checksum' at the end.

The ISO 15765-4 (CAN) protocol uses a message structure that is very similar to that of Figure 3 - see Figure 4, below. The main difference between the two is really only the structure of the header, as CAN does not have distinct bytes, but rather has groups of bits. For this reason, CAN headers are generally known as 'ID bits' and not headers. We use the terms interchangeably, however, as so many people are familiar with our other OBD chips (the ELM320, ELM322, ELM323 and ELM327) which use the term 'header' almost exclusively.

The initial CAN standard stated that there will be 11 ID bits for every message, but that has been expanded and the latest CAN standards now allow for either 11 or 29 bit IDs.

The ELM329 does not normally show anything more than the relevant data bytes unless you turn that feature on with the Headers On command (AT H1).

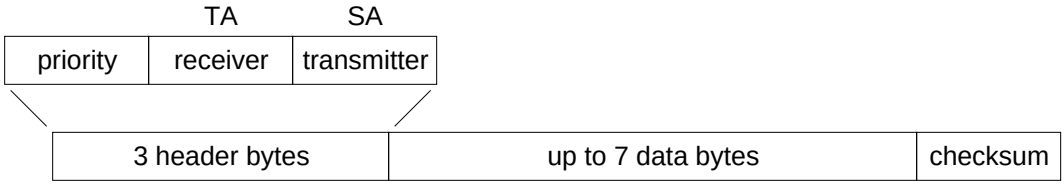


Figure 3. An OBD Message - Initial Protocols

OBD Message Formats (continued)

Issuing it allows you to see the header bytes (ID bits), and other items which are normally hidden such as the PCI byte or possibly the data length code. The current version of the ELM329 does not display the checksum (CRC) information.

It is not necessary to ever have to set the header bytes, or to perform a checksum calculation, as the ELM329 will always do this for you. The header bytes (ID bits) are adjustable however, should you wish to experiment with advanced messages such as those for physical addressing.

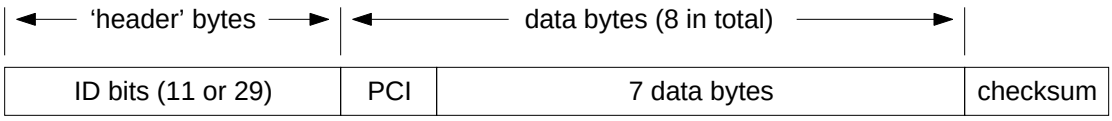


Figure 4. A CAN OBD Message

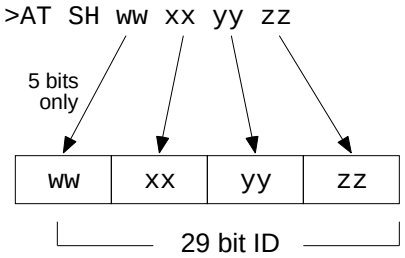
Setting the Header / ID Bits

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 be more specific with your requests.

Accessing most OBDII diagnostics information requires that requests be made to what is known as a a 'functional address.' Any processor that supports the function will respond to the request and, theoretically, many different processors can respond to a single functional request. In addition, every processor (or ECU) will also respond to what is known as their physical address. It is this physical address that uniquely identifies each module in a vehicle, and permits you to direct more specific queries to only one particular module. To direct the queries to a specific address requires changing the values that the ELM329 uses for the header (ID bits).

The ID bits in an ISO 15765-4 header may follow one of two different formats - an 11 bit one, and a 29 bit one. First, consider the 29 bit standard, which has a structure that is very similar to the header structure of older OBD protocols (J1850, etc.).

There are two ways that you may use to define the value that the ELM329 uses for a 29 bit header. The first is to simply provide all of the bits as 4 bytes, or 8 hex digits, using the Set Header command:

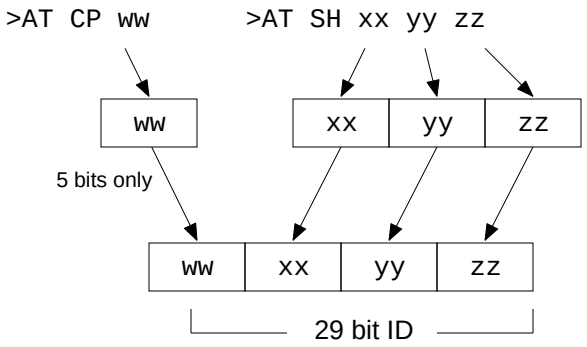


Setting a 29 bit (extended) CAN ID

The ELM329 will ignore the first three bits, leaving 29 that are then used for the messages.

The second way is to change the values in two steps. In this method, the ELM329 splits the 29 bits into a CAN Priority byte and three header bytes. This makes it a little quicker to change only one portion of the header (usually, it is the priority bits that do not change). The two are then combined by the ELM329 into a 29 bit value that it is able to use. To set the header in this way, simply use the CAN Priority and Set Header commands:

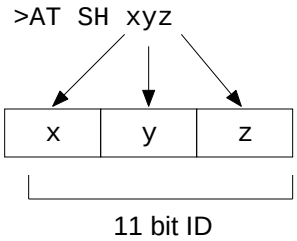
Setting the Header / ID Bits (continued)



Setting a 29 bit (extended) CAN ID

The ISO 15765-4 CAN standard defines each of the above 'byte' values for diagnostics. The priority byte ('ww' in the diagrams) will always be 18 (this is the default value used by the ELM329). The next byte ('xx') describes the type of message that this is, and is set to hex DB for functional addressing, and to DA if using physical addressing. The final two bytes are used in a way that is very similar to other standards – 'yy' is the receiver (or Target Address), and 'zz' is the transmitter (or Source Address). For the functional requests, the receiver is always 33, and the transmitter is F1 (which is very similar to ISO 14230-4).

The other header structure that the CAN standard defines uses an 11 bit ID (and is likely the most common system in use today). The ELM329 uses a special 3 digit version of the Set Header command in order to set these bits:



Setting an 11 bit (standard) CAN ID

In this case, the ELM329 uses the 11 least significant ('right-most') bits of the provided header bytes, and ignores the most significant bit.

The 11 bit ISO 15765-4 CAN standard typically makes functional requests (ID/header = 7DF), but receives physical replies (the header/ID is of the form 7En). With headers turned on, it is a simple matter to see the ID, and so learn the address of the module that is replying. That information can then be used 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
```

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

```
>AT SH 7E0
OK

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

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

When experimenting with different headers, you should be aware that the ELM329 only 'sees' replies that pass through the receive filter. Since the above replies were of the 7En form (which is used by the standard functional OBDII replies), the responses matched the default criteria, and were visible. If the vehicle had replied with something else, then the replies might very well not be visible if you did not take an extra step to define what is to be received. The easiest way to do that is to use the AT CRA (CAN Receive Address) command. In this case, you would only need to say AT CRA 7EX (see the Receive Filtering section on page 42 for more information).

Hopefully this has helped to get you started. As we often tell those that write for help – there is a lot to this, so if you are going to do some serious experimenting with OBD, you should buy the relevant standards.

ISO 15765-4 Message Types

If you are going to be adjusting the header values, you will likely be experimenting with the data bytes as well, so should have some knowledge of the message structures. The ISO 15765-4 standard defines several message types that may be used with diagnostic systems. Currently, there are four of them:

SF - the Single Frame

FF - the First Frame (of a multiframe message)

CF - the Consecutive Frame (" ")

FC - the Flow Control frame

The Single Frame message contains storage for up to seven data bytes in addition to what is known as a PCI (Protocol Control Information) byte. The PCI byte is always the first of the data bytes, and tells how many data bytes are to follow. If the CAN Auto Formatting option is on (CAF1) then the ELM329 will create this byte for you when sending, and remove it for you when receiving. (If the headers are enabled, you will see it in the responses.)

If you turn the Auto Formatting off (with CAF0), it is expected that you will provide all of the data bytes to be sent. For diagnostics systems, this means the PCI byte and the data bytes. The ELM329 will not modify your data in any way, except to add extra padding bytes for you, to ensure that you always send as many data bytes as are required (eight for ISO 15765).

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

Consecutive Frame messages are sent after the First Frame message to provide the remainder of the data. Each Consecutive Frame message includes a single hex digit 'sequence number' that is used to determine the order when reassembling the data. It is expected that if a message were corrupted and resent, it could be out of order by a few packets, but not by more than 16, so the single digit is normally more than adequate. As an example, the serial number for a vehicle is 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
```

The line that begins with 0: is called the 'First Frame'. The length (014) was actually extracted from that line by the ELM329 and printed on the previous line as shown. Following the First Frame line are two Consecutive Frames (that begin with 1: and 2:). To learn more details of the exact formatting, you may want to send a request such as the one above, then repeat the same request with the headers enabled (AT H1). This will show the PCI bytes that are actually used to send these components of the total message.

The Flow Control frame is one that you do not normally have to deal with. When a First Frame message is sent as part of a reply, the ELM329 must tell the sender some technical things (such as how long to delay between Consecutive Frames, etc.) and does so by replying immediately with a Flow Control message. These are predefined by the ISO 15765-4 standard, so can be automatically inserted for you. If you wish to generate custom Flow Control messages, then refer to the 'Altering Flow Control Messages' section, on page 59.

While you will not generally see Flow Control frames while querying a vehicle, you may see them if you are monitoring data requests. If detected, the ELM329 will display such a line with 'FC: ' before the data, to help you with decoding the information.

There is a final type of message that is occasionally reported, but is not supported by the OBD 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 ELM329 always looks for this flag, or for zero data bytes, and may report to you that a Remote Transmission Request was detected while monitoring. This is shown by the characters RTR where data would normally appear, but only if the CAN Auto Formatting is off, or headers are enabled. Often, when monitoring a CAN system with an incorrect baud rate chosen, RTRs may be seen.

Multiline Responses

There are occasions when a vehicle must respond with more information than is able to fit in a single 'message'. In these cases, it responds with several data frames which the receiver must assemble into one complete response. The following shows how this is done with the ISO 15765-4 protocol.

Consider a request for the vehicle identification number, or VIN. This is available from newer vehicles using a mode 09, PID 02 request (but was not initially an OBD requirement, so may not be supported by your vehicle). Here is a typical response that the ELM329 might show:

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

The CAN Formatting has been left on (the default), making the reading of the data easier. With formatting on, the lines begin with a sequence number and then a colon (':') to separate it from the data bytes. CAN systems add this single hex digit (it goes from 0 to F then repeats), to provide an aid for reassembling the data.

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

Looking at the first three bytes of the response, you can see that the first two are the familiar 49 02, as this is a response to an 09 02 request. They can be ignored. The third byte (the '01'), tells the number of data items that are to follow (the vehicle can only have one VIN), and it is not part of the VIN. Eliminating the first three bytes then leaves 17 data bytes which may be used to form the vehicle identification (serial) number. To do this requires first assembling the 17 data bytes in order:

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

The above data values actually represent the ASCII codes for all the characters of the VIN, so the final step is to convert those codes into the actual characters that they represent. ASCII tables are freely available on the web, and may be used to yield the following VIN for the vehicle:

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

From this example, you can see that the format of the data received may not always be obvious. For this reason, a copy of the SAE J1979 (ISO 15031-5) standard would be essential if you are planning to do a lot of work with this, for example if you were writing software to display the received data.

The next example shows how similar messages might occasionally be 'mixed up' in a CAN system. We ask the vehicle for Calibration ID #1 with an 09 04 request and receive the following response:

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

which is quite confusing. The first group (the 013, 0:, 1: group) seems to make some sense (but the number of data bytes do not agree with the response), and the remaining data is also very confusing, as it has two segment twos. It seems that two ECUs are responding and the information is getting mixed up. Which ECU do the responses belong to? The only way to know is to turn on the headers, and repeat your request. Turning the headers on, is simply a matter of sending H1:

```
>AT H1
OK
```

Then you can repeat the request:

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

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

Multiline Responses (continued)

multiline responses.

Looking at the first digits of these responses, you can see that some begin with 7E8, and some begin with 7E9, which are the CAN IDs representing ECU#1 and ECU#2, respectively. Grouping the responses by ECU gives:

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

and

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

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

The information presented here was only meant to provide an overview of how long messages are handled by the ISO 15765 standard. If you do wish to learn more about the actual mechanism, we urge you to purchase a copy of the standard, and study it.

Multiple PID Requests

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

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

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

to which, a typical reply might be:

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

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

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

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

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

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

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

Response Pending Messages

Until the v2.2 update, the ELM329 never looked at the content of any OBD message that it received. Beginning with version 2.2, it now looks at them however, specifically looking for 'response pending' replies.

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

The Response Pending message will always be of the form:

```
7F xx 78
```

where the xx represents the Mode (or Service ID) that was being requested. There is no feedback as to the particular PID requested. As ECUs get busier, these messages are becoming more common, so we now provide support for them in the ELM329.

If bit 2 of PP 2A is set (it is by default), the ELM329 will automatically change the (AT ST) timeout period to 5 seconds for you if it sees a Response Pending message. This is for all CAN protocols by default, but may be refined to only function for ISO 15765 protocols by setting b0 of PP 2A to '1'.

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

Receive Filtering - the CRA command

The ELM329 is always monitoring the CAN data. It retrieves every message from the CAN bus, and then decides whether or not to keep it based on criteria that is established by the ELM329 firmware. This criteria is always initially set to allow OBDII data to pass, but you may change it at any time.

Adjusting the criteria normally takes two steps (see the next section), but there is one AT command that you can use to make life a little easier. It allows setting the address (CAN ID) of messages that you wish to receive, in one simple step.

This command is the 'CAN Receive Address' or CRA command. With it, you can specify a specific address, or a range of addresses that the ELM329 should accept. For example, if the only messages that you wish to see are those that have the CAN ID 7E9, then simply send:

```
>AT CRA 7E9
```

and the ELM329 will set the necessary values so that the only messages that are accepted are the ones with ID 7E9.

If you do not want an exact address, but would prefer to see a range of values, for example all the OBD addresses (those that begin with 7E), then simply use an 'X' for the digit that you do not want the ELM329 to be specific about. That is, to see all messages with CAN IDs that start with 7E (7E0, 7E1, 7E2,..., 7EE, and 7EF), then send:

```
>AT CRA 7EX
```

and the ELM329 will set the necessary values for you.

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

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

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

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

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

and again, the ELM329 makes the necessary changes for you.

Perhaps you do not care which device is sending the information, but you want all messages that start with 18 DA and are being sent to the scan tool. For this, use the character 'X' to tell the ELM329 that you do not care what value a digit has:

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

and the ELM329 takes care of the details for you.

When working with J1939 data, the ELM329 normally formats the data for you, in order to separate the priority from the PGN information. This is usually not a concern when using the CRA command, except when you are trying to filter for a specific priority. For example, you might typically see:

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

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

```
>AT JHF0
OK

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

This more clearly shows the four bytes that need to be defined for the CRA command to be set. For example, to search for all 6 0FEF5's you would actually send the command:

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

In summary then, the CRA command allows you to tell the ELM329 what ID codes to look for, and the letter 'X' may be used in it to represent any single digit that you do not want the ELM329 to be specific about. This is usually selective enough for most applications, but occasionally, there is a need to be specific down to the bit level, rather than to the nibble. For those applications, you will need to program a separate mask and filter, as we show in the next section.

Using the Mask and Filter

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

Internally, the ELM329 configures an 'acceptance filter' with 1's and 0's based on the type of message that it wishes to receive (OBD, J1939, etc.). This pattern is then compared to the ID bits of all incoming messages. If the two patterns match, then the entire message is accepted, and if they do not, the message is rejected.

Having to match all 11 or 29 bits of the ID may be very restrictive in some cases (and would require a very large number of filters for some applications). To allow a little more flexibility in what to accept, and what to reject, a mask is also defined, in addition to the filter. This mask acts just like the type worn on your face - some features are exposed and some are hidden. If the mask has a '1' in a bit position, that bit in the filter must match with the bit in the ID, or the message will be rejected. If the mask bit is a '0', then the ELM329 does not care if that filter bit matches with the message ID bit or not.

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

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

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

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

Putting this together, the filter will have a value:

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

and the mask will have a value:

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

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

```
>AT CF 7E8
OK

>AT CM 7F8
OK
```

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

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

```
18 DA F1 XX
```

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

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

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

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

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

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

Monitoring the Bus

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

To see how your vehicle uses the OBD bus, you can enter the ELM329's 'Monitor All' mode, by sending the command AT MA from your terminal program. This will cause the IC to display any information that it sees on the OBD bus, regardless of transmitter or receiver addresses (it will show all). Note that the ELM329 remains silent while monitoring, so periodic 'wakeup' messages are not sent, and the CAN module does not acknowledge messages (unless CAN Silent Monitoring has been turned off).

The monitoring mode can be stopped at any time by putting a logic low level on the RTS pin, or by sending a single RS232 character to the ELM329. Any convenient character can be used to interrupt the IC as there are no restrictions on whether it is printable, etc. Note that any character that you send will be discarded, and will have no effect on any subsequent commands. The time it takes to respond to such an interrupt will depend on what the ELM329 is doing at the time. The IC will always finish a task that is in progress (printing a line, for example) before printing 'STOPPED' and returning to wait for your input, so it is best to wait for the prompt character ('>') to be sent, or the Busy line to go low, before beginning to send a new command.

Unexpected results occasionally occur if you have the automatic protocol search feature enabled, and you tell the ELM329 to begin monitoring. If the bus is quiet, the ELM329 will begin searching for an active protocol, but it may find something that you were not expecting. The ELM329 may stop searching at one protocol if the baud rate matches, or it might stop at multiples of the actual baud rate (and then report receive errors). If you are testing 'on the bench', the IC might not even find any protocol if the silent mode is enabled (it is by default). Be aware however, that we do not advise setting the silent mode off (AT CSM0) while searching for a protocol to monitor, as the ELM329 may incorrectly interact with the CAN network, and cause problems. In the extreme case, the ELM329 might even have internal problems and report an ERR94.

When monitoring, it is always best if you can select the protocol for the ELM329. If you know that you are looking at a J1939 network, simply tell the ELM329 to set the protocol to A (AT SP A), or if you have an 11 bit, 500kbps ISO15765 system, tell it AT SP 6. The SP command description (page 23) gives a list of all the protocols and their numbers.

If the 'Monitor All' command provides too much information (it does for most CAN systems), then you should restrict the range of data that is to be displayed. The best way to do this is with the CAN Receive Address command (AT CRA).

Perhaps you have an 11 bit system, and only want to see messages that begin with 7. To do that, simply type:

```
>AT CRA 7XX
```

and follow it with an AT MA command. From that point on, all messages that begin with 7 will be displayed (the X's say that you do not care what those other digits are).

Similarly, you might be working with a 29 bit CAN system, and want to see all messages from the engine. If the engine uses address 10, then simply type:

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

and follow it with an AT MA command. From that point on, only messages with IDs that end in '10' will be displayed.

Note that the CRA filter (as well as the CF and CM one) will reduce the amount of information seen with the AT MA command, but there may still be times when the rate that the information is generated by the vehicle far exceeds that which can be handled by the PC connection. In these cases, the internal memory (or 'buffer') fills up more quickly than it is being emptied, and you will see a BUFFER FULL error message. If this is happening, you may wish to consider increasing the baud rate of your connection (see page 47).

Another way to reduce 'BUFFER FULL' errors is by reducing the number of characters that are put into the buffer. You can use AT S0 to eliminate space characters, turn off formatting (AT CAF0) to eliminate 'DATA ERROR' reports, or possibly turn off the headers (AT H0) to eliminate those bytes.

As a final note, the ELM329 can be set to begin 'monitoring all' automatically after power on, if PP 00 is set to the value 00 and is enabled. This only causes an AT MA to be sent, however - there is no facility to automatically provide filtering of the information.

Mixed ID (11 and 29 bit) Sending

Users often ask if the ELM329 can send CAN data other than that used for OBD. It is certainly able to do that - all it takes is an understanding of the way that messages are formed and sent by the chip. The ELM329 also has two special instructions that make it even easier to send any CAN message at any time.

When you provide a mode and PID for sending, the ELM329 puts that request within a message structure just like that shown in Figure 5. For this example, we've assumed an ISO 15765-4 protocol with an 11 bit ID, and an 01 05 (coolant temperature) request. Notice that the two data bytes remain intact - they are not altered in any way.

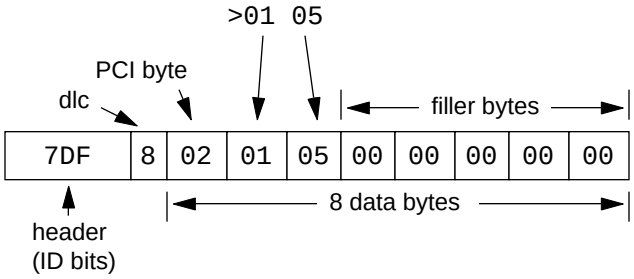


Figure 5. ISO 15765-4 Request

Every ISO 15765 message requires that there be a special data byte (called a PCI byte) in the first position. The ELM329 automatically adds this byte for you, if automatic formatting is turned on (it is by default). If you do not want this byte added, simply turn the formatting off, with the AT CAF0 command.

This protocol also requires that all messages have 8 data bytes (the CAN protocol allows 0 to 8). If, as above, the message is less than 8 bytes long, the ELM329 will add extra 'filler' bytes for you in order to make the length 8 bytes. If you do not want to send 8 bytes, use the AT V1 command to allow the messages to be variable in length.

When you turn the formatting off, and allow variable data lengths, it's not as easy to send standard ISO 15765 requests (but not impossible). If you had done as above and sent AT CAF0, followed by AT V1, then wanted to request the coolant temperature, all you need to do is provide the 8 data bytes yourself:

```
>02 01 05 00 00 00 00 00
```

You do not always have to use the CAF0 and V1 commands in order to send arbitrary data using the ELM329, however. An alternative is to define your own protocol, using the Programmable Parameters for the User protocols. Actually, protocol B provides a quick way to do this:

```
>AT PB C0 02
OK
```

is all that is needed to set up a 250 kbaud protocol that has an 11 bit ID, variable data length send, and no formatting.

The above discussion showed how to set options to modify existing protocols, or to create a new one that is able to send any data for you. The ELM329 also provides two special commands that add this flexibility at any time, to any protocol.

If you were in the original situation (Figure 5), and wished to send the four data bytes '11 22 33 44' with no formatting (ie no PCI byte) or filler bytes, then all you need do with the ELM329 is send:

```
>.11 22 33 44
```

Notice the single dot ('.') out front, which tells the ELM329 to use the 11 bit ID. If you had used two dots (':') as follows:

```
>:11 22 33 44
```

then the ELM329 would have sent the message using the 29 bit ID (you can set the value with the AT SH command).

Note that messages with 11 or 29 bit IDs can be sent at any time using these two commands, no matter what the current protocol uses. The only restriction is that the current protocol must be active - that is, you must have been sending requests and receiving replies (so the ELM329 knows what the baud rate and other settings should be).

The '.' and ':' commands always use the currently defined headers for sending. If you wish to send with something different, then the standard AT SH command should be used to set either the 11 bit, or the 29 bit header. (One of these headers will also affect the current protocol though, as there is no facility to define more than one 11 bit, or one 29 bit header.)

Note that the '.' and ':' commands also never modify your data in any way - they do not add any formatting bytes, and they do not add filler bytes. If you want a message that has 8 data bytes, then you must provide all 8 data bytes.

Restoring Order

There may be times when it seems the ELM329 is out of control, and you will need to know how to restore order. Before we continue to discuss modifying too many parameters, this seems to be a good point to discuss how to 'get back to the start'. Perhaps you have told the ELM329 to monitor all data, and there are screens and screens of data flying by. Perhaps the IC is now responding with 'NO DATA' when it did work previously. This is when a few tips may help.

The ELM329 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 for the current line, then stop, print a prompt character, and wait for your input. The stopping may not always seem immediate if the RS232 send buffer is almost full, though – you will not actually see the prompt character until the buffer has emptied, and your terminal program has finished printing what it has received.

There are times when the problems seem more serious and you don't remember just what you did to make them so bad. Perhaps you have 'adjusted' some of the timers, then experimented with the CAN filter, or perhaps tried to see what happens if the header bytes are changed. If you have been experimenting with CAN filters and are suddenly seeing 'NO DATA' responses, this can usually be fixed by resetting the filters. Simply send:

```
>AT CRA
OK
```

and the filter and mask will be reset to the default values.

If you problem is more involved than this, then all of the settings can be reset by sending the 'set to Defaults' 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 ELM329 will close the current session and then change the protocol to the default one, exactly as instructed.

If the AT D does not bring the expected results, it may be necessary to do something more drastic - like resetting the entire IC. There are a few ways that this can be performed with the ELM329. One way is to simply remove the power and then reapply it. Another way that acts exactly the same way as a power off and then on is to send the full reset command:

```
>AT Z
```

It takes approximately one second for the IC to perform this reset, initialize everything and then test the four status LEDs in sequence. A much quicker option is available with the ELM329, however, if the led test is not required – the 'Warm Start' command:

```
>AT WS
```

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

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

```
>AT PP FF OFF
```

which should disable all of the changes that you have made. Since some of the Programmable Parameters are only read during a system reset, you may have to follow this command with a system reset:

```
>AT Z
```

after which, you can start over with what is essentially a device with 'factory settings'. There may be times when even this command is not recognized, however. If that is the case, you will need to use the hardware method of turning the PPs off. See the section on 'Programmable Parameters' (pages 69 to 70) for more details.

Using Higher RS232 Baud Rates

The RS232 serial interface has been maintained throughout the ELM OBD products, largely due to its versatility. Although the ELM329 only offers a serial 0 to 5V signal level, you may connect that to a large number of interfaces. Traditional RS232 interfaces such as those shown in Figures 11 and 12 (in the Example Applications section) may be used to connect to older computers, microprocessors and PDAs. Alternatively, a USB converter, a Bluetooth module, or possibly ethernet or wifi interfaces may be used to connect to many more devices and systems. It is simply one of the most versatile interfaces available.

Many circuits have been built using the ELM329 integrated circuit and a serial interface. Typical interfaces such as those shown in Figures 11 and 12 (in the Example Applications section) work very well, but there are many other alternatives that work just as well. For most applications all that is required is the default 38.4 kbps data rate, and the circuit of Figure 11 works very well for this. If pushed beyond 57.6 kbps, however, it will likely begin to have problems. The circuit of Figure 12 offers a solution if you are looking to extend your data rate up to 250 kbps (if you use a MAX3222E, but a MAX3222 is limited to 120 kbps).

A standard RS232 interface needs large voltage swings, which are difficult to maintain at high data rates (as there are large cable capacitances to contend with). For this reason, if you wish to operate at rates higher than about 230 kbps, you should look at alternatives.

One alternative is a direct connection to a microprocessor, which we discuss in the Microprocessor Interfaces section. Another alternative is to use a USB interface.

The USB interface is capable of very high data transfer rates, certainly much higher than the ELM329 is capable of. Several manufacturers offer special 'bridge' circuits that simplify connecting a serial device (such as the ELM329) directly to the USB bus. Examples are the FT232R or the DB9-USB module from Future Technology Devices (see their web site at http://www.ftdichip.com/) or the CP2102 from Silicon Labs (http://www.silabs.com/). We show the DB9-USB module connected in Figure 9, while the CP2102 is shown in Figure 13. If planning to use the higher baud rates, a USB interface should be seriously considered.

No matter what type of connection you use to bring your ELM329 data out, the ELM329 will see it as a serial connection. As shipped, the ELM329 can be used at data rates of either 9600 baud, or 38400 baud (the voltage level at pin 6 during power up or reset determines the rate used). While the 9600 baud rate is not adjustable, the 38400 one is. There are two ways that the rate can be changed – either permanently with a Programmable Parameter, or temporarily with an AT command.

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

If you store a new value in PP 0C, then enable it, and if pin 6 is at a high level during the next powerup, your stored rate will become the new data rate. As an example, perhaps you would like to have the ELM329 use a baud rate of 57.6 kbps, rather than the factory setting of 38.4 kbps. To do this, you will need to determine the required value for PP 0C, store this value in PP 0C, and then enable the PP.

The value stored in PP 0C is actually an internal divisor that is used to determine the baud rate (the baud rate will be 4000 kbps divided by the value stored in PP 0C). To obtain a setting of 57.6, a baud rate divisor of 69 is required (4000/69 is approximately 57.6). Since 69 in decimal is 45 in hexadecimal, you need to tell the ELM329 to set the value of PP 0C to 45. The following command is used for that:

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

To use the value stored in PP 0C requires that it be enabled as well:

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

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

If you are designing your own circuitry, you will know what speed your interface is capable of, and can simply assign a value to PP 0C. Software developers will not usually know what hardware is to be connected, however, so will not know what the limitations are. For these users, we have provided the BRD command.

This command allows a new baud rate divisor to be tested, and then accepted or rejected depending on the results of the test. The chart shown here helps to

Using Higher RS232 Baud Rates (continued)

explain how the command works.

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

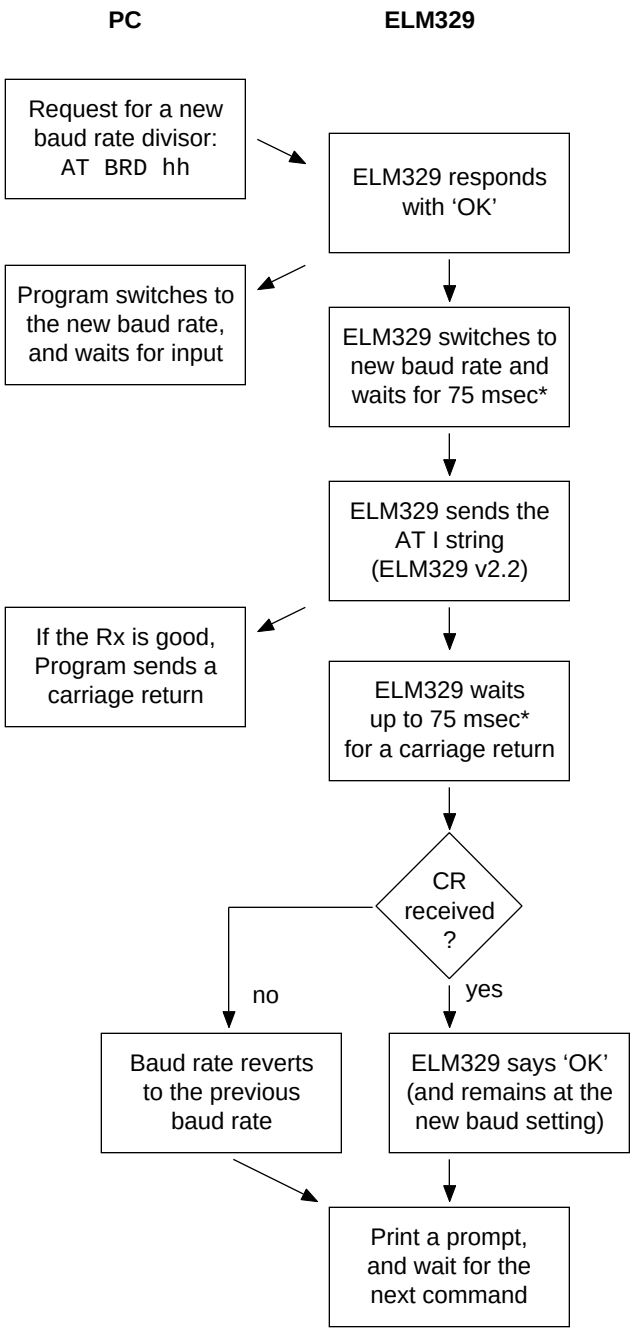
```
AT BRD 45
```

and the ELM329 would respond with 'OK'. After it sees the 'OK', the PC should switch to the new data rate of 57.6 kbaud. Note that no prompt character follows the ELM329's 'OK' reply - it is followed only by a carriage return character (and optionally, a linefeed character).

Having sent an 'OK', the ELM329 also switches to the new (proposed) baud rate, and then simply waits a predetermined time (nominally 75 msec). This period is to allow the PC sufficient time to change its baud rate. When the time is up, the ELM329 then sends the ID string (currently 'ELM329 v2.2' if PP 08 = FF) to the PC at the new baud rate, followed by a carriage return and a linefeed (if enabled). It then waits for a response.

Knowing that it should receive the ELM329 ID string, the PC software compares what was actually received to what was expected. If they match, the PC responds with a carriage return character, but if there is a problem, the PC sends nothing. The ELM329 is meanwhile waiting for a valid carriage return character to arrive. If it does (within 75 msec), the proposed baud rate is retained, and the ELM329 says 'OK' at this new rate. If it does not see the carriage return, the baud rate reverts back to the old rate. Note that the PC might correctly output the carriage return at this new rate, but the interface circuitry could corrupt the character, and the ELM329 might not see a valid response, so your software must check for an 'OK' response before assuming that the new rate has been accepted.

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

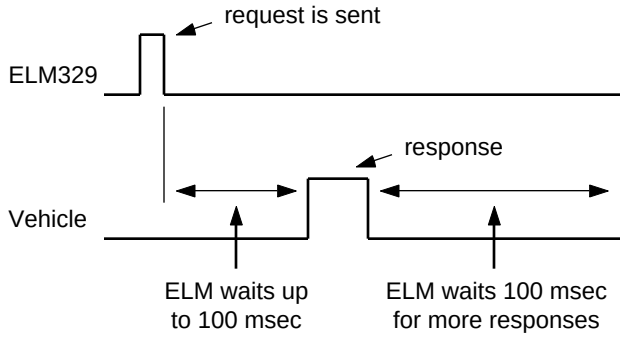


* the 75 msec time is adjustable with the AT BRT hh command

Setting Timeouts - the AT ST and AT AT Commands

Users often ask about how to obtain faster OBD scanning rates. There is no definite answer for all vehicles, but the following information may help with your understanding of how the AT ST and AT AT settings are used by the ELM329.

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



The ELM329 sends a request then waits up to 100 msec for a reply (the standard requires 50 msec). If no reply arrives in that time, an internal timer stops the waiting, and the ELM329 prints 'NO DATA'. If a reply has been received, the ELM329 must wait to see if any more replies are coming (and it uses the same internal timer to stop the waiting if no more replies arrive). While all replies should be received within 50 msec, the 100 msec setting ensures that a response is not missed.

As an example, consider a vehicle that responds to a query in 10 msec. With the ST timeout set to 100 msec, the fastest scan rate possible would only be about 9 queries per second (it's 10 + 100 msec per response). Changing the ST time to about 40 msec would more than double that rate, giving about 20 queries per second. Clearly, if you were to know how long it takes for your vehicle to reply, you would be able to improve on the scan rate, by adjusting the ST time.

It is not easy to tell how fast a vehicle replies to requests. For one thing, requests all have priorities assigned, so responses may be fast at some times, and slower at others. Even when a response begins, different frames within a multi-frame response can have very different delays. The physical measurement of the time is not easy either - it requires expensive test equipment just to make one measurement. To help with this, the ELM329 includes a feature called 'Adaptive Timing'.

Adaptive Timing actually measures the response times for you, averages several readings, and then adjusts the AT ST time to a value that should work for most situations. It is enabled by default, but can be disabled with the AT0 command should you not agree with what it is doing (there is also an AT2 setting that is a little more aggressive, should you wish to experiment). For 99% of all vehicles, we recommend that you simply leave the settings at their default values, and let the ELM329 make the adjustments for you.

OK - the ELM329 is able to measure times, and suggest a setting for the AT ST time, but the IC still has to wait after receiving a reply to see if any more are coming. Surely there has to be a way to eliminate that final timeout, if you know how many responses to expect? There is a way - by telling the ELM329 how many messages to receive.

If you wish to make a request, and know how many responses there should be, simply add that response count as a single digit after your request. For example, if you know that two ECUs will respond to an 01 00 request, then send:

```
>01 00 2
```

The ELM329 will send the 01 00 request, and will return to the prompt state immediately after the second response is received (or after the ST timer times out if the response does not arrive). In this way, every response is shortened by that ST time. This can increase the polling rate considerably for most vehicles (many users report achieving 50 or more samples per second).

In general, you do not know how many ECUs will respond to a request, so this feature is best used by software that can query the vehicle to determine the number of responses that will be coming, store that value, and then use it to set the responses digit for subsequent requests.

SAE J1939 Messages

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

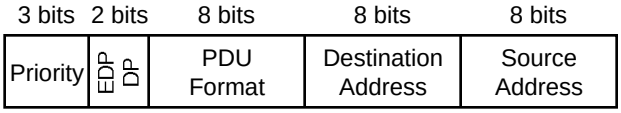
The following will discuss a little of how data is transferred using the J1939 standard. Considerably more information is provided in the Society of Automotive Engineers (SAE) standards documents, so if you are going to be doing a lot of work with J1939, it may be wise to purchase copies of them. At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

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

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

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

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



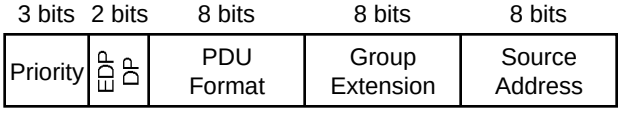
PDU1 Format

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

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

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

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



PDU2 Format

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

SAE J1939 Messages (continued)

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

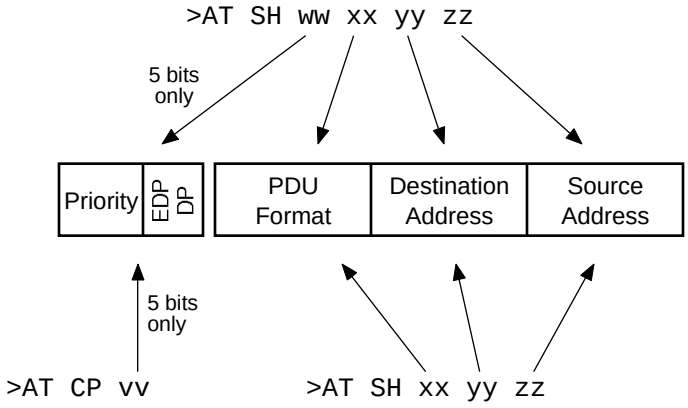
To summarize, PDU1 format frames are sent to a specific address, and PDU2 frames are sent to all addresses. To further complicate matters, however, PDU1 frames may be sent to all addresses. This is done by sending the message to a special 'global address' which has the value FF. That is, if you see a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

The J1939 recommended practices document provides a list of addresses that should be used by devices. It is particularly important to adhere to this list with the ELM329, as the IC uses a fixed address method and is not able to negotiate a different one, per J1939-81. OBD Service Tools should use either F9 or FA as their address (the ELM329 uses F9). If you wish to change this, you can use the AT TA (tester address) command, or simply define it with the header.

Headers (ID bits) are assigned using the Set Headers command. All 29 bits may be assigned at once, as shown below, or they may be set in two steps (using the AT CP and AT SH commands). The two step process may be useful if you only want to change some of the ID bits rather than all of them (the priority values only rarely change, so it's often quicker to use the three byte AT SH command, and leave the priority bits unchanged). Note that whether you use the AT CP, or the four byte AT SH command, the three most significant bits are ignored by the ELM329.

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

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



Setting the J1939 CAN ID

Using J1939

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

To begin, you will need to configure the ELM329 for J1939 operation, at the correct baud rate. Protocol A is predefined for J1939 at 250 kbps, which is what most applications require. To use protocol A, send:

```
>AT SP A
```

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

```
>AT PB 42 01
```

then send:

```
>AT SP B
```

to select it. Note that this setting will not be maintained if the IC is reset, so if you may want to make it a permanent setting, by storing the values in PP 2C and 2D. To do this, first, set the values with:

```
>AT PP 2C SV 42
```

and:

```
>AT PP 2D SV 01
```

then turn them on to enable the changes:

```
>AT PP 2C ON
```

and:

```
>AT PP 2D ON
```

Once the protocol is set, then you are ready to go. There is no need to adjust anything else (timing, etc.) as that is all done for you.

If you do wish to adjust the timing, you should be aware that the ELM329 provides the ability to extend the AT ST time by switching a x5 timer multiplier on and off (see the STM5 and JTM5 commands). This may be required if requesting data that will have a multiline response while other multiline broadcasts are already being sent. If you do not seem to be seeing a response that you expect to see, try extending the timeout value and seeing if it then appears. The timer multiplier may be restored to normal with either AT JTM1 or AT STM1.

Once the J1939 protocol has been selected, the ELM329 is ready for a command. The first one that we will use is called a DM1 or ‘diagnostic message 1’, which provides the currently active diagnostic trouble codes. DM1 is one of more than 50 predefined diagnostic messages, and is special in that it is the only one that is broadcast continually over the bus at regular intervals. The ELM329 has an AT command that can be used to obtain the DM1 trouble codes:

```
>AT DM1
```

If you are connected to a vehicle, you should now see messages printed at one second intervals. If you are only connected to a single device (for example, with a simulator on the bench, or to a device with a single CAN data port), you may see data with <RX ERROR printed beside it. This is because the receipt of the data is not being acknowledged by any device on the bus (certainly not the ELM329, as it is by default a completely silent monitor). See our ‘AN05 - Bench Testing OBD Interfaces' application note for more information on this, and some advice on what to do. If you are not connected to a vehicle, and are having trouble receiving data, try sending:

```
>AT CSM 0
```

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

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

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

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

Using J1939 (continued)

```
>AT MP 00FECA
```

Note that the ELM329 requires that you send hex digits for all data, as shown above (and as used by all other protocols). Many of the PGN numbers are listed in the J1939 standard as both a decimal and a hex number, so be careful to choose the hex version.

You will likely find in your testing that the PGNs you encounter often begin with a 00 byte as above. To simplify matters for you, the ELM329 has a special version of the MP command that will accept a four digit PGN, and assumes that the missing byte should be 00. An equivalent way to ask for 00FECA is then:

```
>AT MP FECA
```

which is a little more convenient. Please note that the MP command is very similar to the MA command, except that it is able to process multiline responses. If you are simply interested in receiving single line broadcast messages, then using the CRA and MA commands may be an option.

Just as the ELM329 allows the number of ISO 15765 responses to be specified when a request is made, it also allows you to specify the number of messages to retrieve when monitoring for PGNs. It is done in the same way - for example, to specify only two responses for the MP FECA command, send:

```
>AT MP FECA 2
```

This saves having to send a character to stop the flow of data (as you would with AT DM1), and also is very convenient when dealing with multiline messages. While the standard OBD requests allow you to define how many frames (ie lines) of information are to be printed with a similar single digit, the single digit with the MP command actually defines how many complete messages to obtain. For example, if the DM1 message is 33 lines long, then sending AT MP FECA 1 will cause the ELM329 to show all 33 lines, then stop monitoring and print a prompt character.

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

```
>AT H1
```

A single response to FECA might then look like:

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

Notice that the ELM329 separates the priority bits from the PGN information. The ELM329 also uses only one digit to represent the two extra PGN bits, both of which may seem unusual if you are used to different software. We find this a convenient way to show the actual J1939 information in the header. Note that version 1.0 of the ELM329 always assumed that the Extended Data Page (EDP) bit was 0 when printing formatted output as shown above. Beginning with v2.0, the ELM329 now displays both the EDP and the DP bits (the EDP should always be 0 for J1939, but other protocols do use this bit).

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

```
>AT JHF0
```

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

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

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

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

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

```
>00FEEE
```

to which you might receive:

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

if the headers were on. Note that if you request a PGN that is already being broadcast, you may very well receive many replies, as the ELM329 configures itself

Using J1939 (continued)

to receive anything that is related to the PGN requested.

If you are familiar with the J1939 standard, you will be aware that it actually specifies a reverse order for the sending of the data bytes of a PGN request. That is, the data bytes for the above request are actually sent as EE FE 00, and not as 00 FE EE. Since it can be very confusing to have to reverse some numbers and not others, the ELM329 automatically handles this for you, reversing the bytes provided. In this way, you can directly request PGNs using numbers as they are written on the page (if they are written as hex digits), and the ELM329 will make it work for you. If you do not want the ELM329 to alter the byte order, the feature can be disabled (by sending an AT JS command).

The ELM329 always assumes that when you start making requests of this type, you do not know what devices are connected to the J1939 bus. That is, by default the ELM329 sends all requests to the 'global address' (ie all devices), and then looks for replies. Often, this works well, but J1939 devices are not required to respond to such general inquiries, and may not if they are busy. For this reason, it is usually better to direct your queries to a specific address, once it is known.

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

By default, the ELM329 uses 6 0EAFF F9 for the ID bits of all requests (or 18 EA FF F9 if you prefer). That is, it uses a priority of 6, to make a request (EA) to the global address (FF) by the device at F9 (the scan tool). Since you only wish to alter the EAFF F9 portion of the ID bits and not the priority, you may do this with the three byte set header command:

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

As an aside, note that hexadecimal EA00 is the same as decimal 59904. For this reason, messages with EA for the PDU Format value are often referred to as PGN 59904 requests.

After making the above change, all data requests will be directed to the engine, so don't forget to change the headers if you wish to again make global requests. Note that the AT SH command allows you to change the source (or tester) address at will, so be careful with this as addresses should really be negotiated using the method described in J1939-81 and you might conceivably choose an address that is already in use. The current version of the ELM329 does not support J1939-81 address negotiation, so can not obtain an address for you.

Once the ELM329 has been configured to send all messages to address 00, repeat the request:

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

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

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

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

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

All responses to a request are printed by the ELM329, whether they are a single CAN message, or a multisegment transmission as defined by the transport protocol (J1939-21). If the responses are multisegment, the ELM329 handles all of the negotiation for you. As an example, a multisegment response to a DM2 request might look like this:

```
>00FECB
012
```

Using J1939 (continued)

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

if the headers are on, and would appear as:

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

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

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

The one line that shows the total number of data bytes is actually called a 'Connection Management' or 'TP.CM' message. It has a specific format, but the only bytes that are typically relevant are those that provide the total message size in bytes. In order to see the other bytes, you must turn CAN Auto Formatting off (AT CAF0), and then repeat the request.

This has been a brief description of how to use the ELM329 in a typical J1939 environment. If you can monitor for information, make global requests as well as specific ones, and receive single or multiframe responses, then you have the tools necessary to at least diagnose most vehicle problems.

The FMS Standard

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

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

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

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

The FMS standard is completely open, and still evolving (as of this writing, the Bus and Truck standards have been combined into one document – the latest update was version 4.00, dated October 13, 2017).

Visit the FMS Standard web site for more information:

http://www.fms-standard.com

The NMEA 2000 Standard

We are occasionally asked about support for the NMEA 2000 marine standard. Elm Electronics does not provide specific support for this protocol, but our ELM329 integrated circuit is very capable of working with the protocol.

While the physical connectors may look quite a bit different than those used for J1939, the CAN interface and the data format is almost identical to that of the J1939 standard. NMEA 2000 uses a 250 kbps data rate, so the easiest way to get started is to select the ELM329's predefined protocol A. This is done with the set protocol to A command:

```
>AT SP A
```

When you are finished and want to use the ELM329 for standard OBDII protocols, don't forget to send the AT SP 0 command to reset it.

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

```
>AT MP 1F200
```

as the ELM329 actually interprets that command as an AT MP 1F20 0 request (ie. Monitor for PGN 1F20, and get 0 replies). To monitor for PGN 1F200, you must send the full 6 digit address:

```
>AT MP 01F200
```

If you keep the above in mind, the ELM329 will prove to be a handy tool to use while experimenting with NMEA 2000. It does have a couple of limitations that must be kept in mind, though. As mentioned with J1939, it is not capable of address negotiation. Also, the ELM329 does not currently support the Fast Packet protocol, which may be an issue for some users.

For more information on the NMEA 2000 standard, visit the National Marine Electronics Association web site:

http://www.nmea.org

Sending UDS Messages

As vehicles continue to become more and more complex, the need for more and better diagnosis increases. The older OBD standards that defined only a few modes and single byte PIDs no longer provide enough unique identifiers to be truly useful. In order to address this expanding need, and the ways in which the information might be obtained, the Unified Diagnostic Services (UDS) standard was developed. UDS is defined by the ISO 14229 set of standards, with the ISO 14229-1 and ISO 14229-3 (UDSonCAN) documents of primary interest here.

It is the UDSonCAN version that the ELM329 supports, since it is essentially the ISO 15765 protocol with more modes. Rather than call them 'modes' however, the first byte is called a Service ID (SID) byte – which is much more generic.

One of the first SIDs that you might encounter is SID 10, which controls the type of interactions that are allowed to occur between the tester and the vehicle. The most common is the default session (01) which allows basic data transfer and control. The ECU normally is in this default mode and will revert to it, should one of the other types 'time out'. The other session types are used for programming, extended diagnostics, etc. and are discussed in ISO 14229-1.

You should not have to initiate a default session, but if you need to, this can be done by sending the Service ID 10 command, with diagnosticSessionType equal to 01. To do this, first make sure that you are connected to the vehicle and that the protocol is 'active'. This is most easily done by sending 0100 and obtaining a reply. Turn the headers on too, so that you know which ECU is responding:

```
>AT H1
OK

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

The response to 0100 shows that the vehicle does use the ISO 15765 protocol, and that a connection has been made with the controller which uses the 7E8 ID. Once you have this connection made, you may then use the Session Control SID (10) to open a default UDS session with diagnosticSessionType 01:

```
>10 01
7E8 06 50 01 00 32 01 F4
```

The bytes returned are what you would expect for an ECU reply. The ID bits (7E8) and the PCI byte (06) are standard for ISO 15765, the 50 is what you would expect in response to a 10 request (the SID + 40) and the 01 is the PID/DID that you had requested. The last four bytes may or may not be present in your reply, as early implementations did not require these in all responses. The first pair represents the maximum time that the ECU can take to respond (hex 0032 x 1 msec = 50 msec), and the maximum time for a subsequent response after a 'Response Pending' reply (hex 01F4 x 10 msec = 5 sec).

When the 10 01 command was sent, the ID bits were still set to the (default) 7DF functional value, but many UDS commands require that they be directed to the physical address for the ECU. From the ISO 15765 standard, the physical address for an ECU that responds with 7E8 is 7E0, so in order to send messages to that address, we need to change the header (ID) value used by the ELM329. Simply send the Set Header command to do this:

```
>AT SH 7E0
OK
```

Now, we can send other SIDs, such as 22 (which is used to read data by the identifier). ISO 14229 defines many DataIdentifiers (or DIDs), but the F4, F6 and F8 are the ones of interest here as they may be used to obtain mode 01, 06 and 09 data, respectively. For example, to read coolant temperature, you would send:

```
>22 F4 05
```

while reading multiple DIDs with one command is simply a matter of providing both:

```
>22 F4 0C F4 0D
```

As always, we have provided enough information here to get you started. If you wish to experiment further, you may search on the web for help, and purchase the relevant standards. All ISO 14229 documents are available from the International Organization for Standardization (www.iso.org).

Occasionally, UDS applications will require switching to other session types. Those that are not the default type will require periodic messages in order to keep the session active. The following section discusses how to define and enable these periodic messages.

Periodic (Wakeup) Messages

Some applications require that there be periodic messages sent by the test equipment (scan tool) in order to maintain a connection. If these messages do not arrive in a timely fashion, the ECU will either revert to a default mode of operation, or close the connection and go into a low power 'sleep' mode. In order to stop the ECU from doing this, you will need to send what we term as 'wakeup' messages. Some texts also refer to these as CAN periodic messages.

The ELM329 does not send wakeup messages by default, even if you set PP 23 to 01 or 02. There are several other conditions that must be met in order for sending to be enabled.

First, a defined CAN protocol must be selected (i.e. the IC is not set to protocol 0) and the protocol must be in the 'normal' mode of operation. That is, it must be in the mode where you send and receive messages, not in one of the 'monitoring' modes (as entered with AT MA, DM1, or MP), unless you have disabled the silent mode with AT CSM0.

If the default wakeup mode is not set to 01 or 02 by PP 23, then you must select either WM1 or WM2 with the AT commands, and the selected protocol must be made active – see What is an Active Protocol? (page 34). This is normally sufficient to start the sending of wakeup messages. If you do not want them to start at this point, you may wish to use AT W0 or AT SW 00 to block them.

Typically, you will set the wakeup variables (ID bits, data bytes and timing) before making a protocol active. That way your adjustments don't cause the periodic messages to be sent until you are ready for them (and you won't need to use AT W0 or AT SW 00 to block them).

Once enabled, the ELM329 sends the following periodic message by default:

```
7DF 01 3E 00 00 00 00 00 00
```

Note that this default wakeup message uses an 11 bit ID, and sends 8 data bytes. This message will be sent, even if the current protocol uses a 29 bit ID. If you wish to send a 29 bit ID, then you will need to define that ID with the 'Wakeup Header' command. For example, with:

```
>AT WH 18 DB 33 F1
```

From that point on, all of the wakeup messages will be sent with this 29 bit header (the ELM329 always uses the last ID that was defined using AT WH). Of course, the above header is only an example - you may define any values that you wish for the ID bits.

Setting the actual content of the wakeup message is done with the Wakeup Data (AT WD) command. The ELM329 does not format the data provided in any way, and it does not pad the length out to 8 bytes. Whatever you provide will be used exactly as you present it. For example, sending:

```
>AT WD 01 3E
```

with the 11 bit example from above, will result in the following message being sent by the ELM329:

```
7DF 01 3E
```

The difference here is that the data length is no longer 8 bytes - it has been set to two bytes, as that is what was provided (so if you want the message to use 8 bytes, you need to provide all 8 bytes). UDS requires 8 bytes, typically 02 3E 80 00 00 00 00 00.

Once you have the wakeup header and data set as you want them, you only need to select the mode. Whether you use 1 or 2 depends on the protocol (it is typically mode 2 for physical addressing, and mode 1 for functional, but you need to verify this). To set the mode, simply use the Wakeup Mode command:

```
>AT WM 2
```

Enabling Wakeup Mode 2 may result in the wakeup messages being sent if the protocol is active, and the other conditions have been met. Again, you may wish to use AT SW 00 or AT W0 to control the sending.

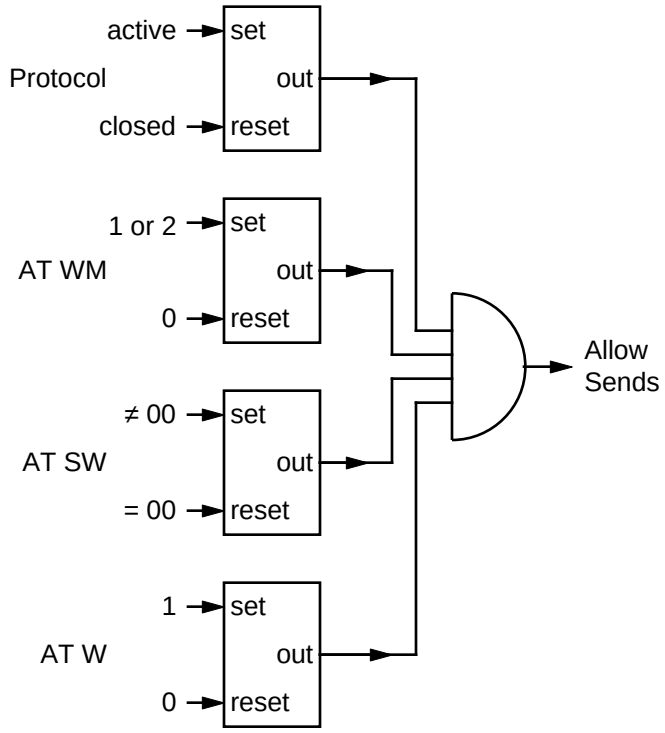
The time interval between the wakeup messages can be adjusted in 20.48 msec increments using the AT SW or AT WT commands. Simply provide the setting that you require as two hex digits - for example, a setting of:

```
>AT SW 92
```

will result in a timer setting of about 3 seconds (92 hex is 146 decimal, giving 2.99 seconds). The default timer setting is 62 (98 decimal) giving 2 seconds.

The wakeup time as set with AT SW or AT WT is actually a value used to compare to an internal timer/counter. For this reason, caution should be used if adjusting this setting on-the-fly as it will affect the current period. The timer/counter will be reset

Periodic (Wakeup) Messages (continued)



whenever you select the wakeup mode, or when you use AT SW (but AT SW only resets it if the wakeups are not being sent at the time).

The setting of the wakeup message header bits, data bytes, and period are very straight-forward, but how the sending of these messages is enabled may lead to some confusion. The diagram shown at left might be of help with this. It shows the four separate conditions controlling the outputs of set-reset flip flops. All four outputs must be set in order for all the inputs to the and gate to be high, which in turn enables the sending of the periodic messages. Default values are Protocol closed, WM0, SW 62 (i.e. ≠ 00), and W1. Note that previous to v2.2, the ELM329 did not support the AT W instruction, so its logic diagram would only have the 3 flip-flops.

Finally, as mentioned previously, it is also possible to have periodic messages sent while monitoring if all of these four conditions have been met, and you have also chosen to disable the default silent mode with AT CSM0. This is not a normal situation however, so is not shown on the diagram

Altering Flow Control Messages

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

To send one of these multi-line messages, the transmitter in a CAN system will send a 'First Frame' message, and then wait for a reply from the receiver. This reply, called a 'Flow Control' message contains information concerning acceptable message timing, etc., and is required to be sent before the transmitter will send any more data. For ISO 15765-4, the type of response is well defined, and never changes. The ELM329 will automatically send this ISO 15765-4 Flow Control response for you as long as the CAN Flow Control option is enabled (CFC1), which it is by default.

The ELM329 allows you to customize how it responds when it needs to send a Flow Control message, by changing the Flow Control 'modes'. You can leave it as a fully automatic response (mode 0), can provide only the data bytes that you want sent (mode 2) or can define both the header (ID bits) and the data bytes (mode 1).

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

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

This will immediately restore the responses to their default settings.

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

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

You must set the headers and data first:

Altering Flow Control Messages (continued)

```
>AT FC SH 7E8
OK

>AT FC SD 30 00 00
OK
```

and then you can set the mode:

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

From this point on, every First Frame message received will be responded to with the custom message that you have defined (7E8 00 11 22 in this example). Note that the number of bits in the flow control header does not have to match the number in the active protocol (you may define a 29 bit header for 11 bit systems, etc.)

The third mode currently supported allows the user to set the data bytes which are to be sent. The ID bits (header bytes) in this mode are set to those which were received in the First Frame message, without change. To use this mode, first define your data bytes, then activate the mode:

```
>AT FC SD 30 00 00
OK

>AT FC SM 2
OK
```

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

The following chart summarizes the currently supported flow control modes:

FC Mode	ELM329 Provides	User Provides
0	ID Bits & Data Bytes	no values
1	no values	ID Bits & Data Bytes
2	ID Bits	Data Bytes

Flow Control Modes

Note that the ELM329 will only send Flow Control messages if the current data format is ISO 15765-4.

Sending Multiline Messages

We occasionally receive questions concerning the sending of multiline (i.e. multiple frame) messages. This is not very difficult if you understand the relevant standards and are able to send your message frames under program control.

Multiline messages begin with a First Frame message. This contains the total number of bytes (in all the frames) that are to be sent, and the first six data bytes. After receiving a First Frame message, an ECU will respond with a Flow Control message that you need to decode in order to be sure of the readiness of the ECU, and the message timing requirements. Typically, there will be no delays of more than 50 msec (the P2CAN time) between messages, but an ECU may specify a minimum time that is greater than this. Also, OBDII typically sends messages as one block of data, but the ECU that you are connected to may request that you split the data differently.

After receiving the Flow Control message, you are are usually able to send the remaining data in Consecutive Frames. These have a segment number associated with each that needs to be incremented, and they contain seven of the data bytes. When you reach the final Consecutive Frame, if there are unused data byte positions after your data, you must fill them with padding bytes (as the total number of bytes in the CAN message must be eight).

Several of our customers have voiced a need for the ELM329 to provide 'better' support for the sending of multiline messages. The use of these messages are for applications well beyond simply reading trouble codes, and VINs, however. We do not intend to support this, and advise great caution if you are planning to experiment with it. Obtain the relevant standards, study them, understand them and have a plan if things go wrong. Elm Electronics can not support you if you go down this road.

Using CAN Extended Addresses

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

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

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

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

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

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

As an example, we will set the ELM329 to handle the above messages. We know that the messages use 11 bit IDs with ISO 15765 formatting, and let us assume a baud rate of 50kbps. The command to configure protocol B for this is:

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

Next, tell the ELM329 to receive all messages that have an ID of 7C0:

```
>AT CRA 7C0
```

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

```
>AT SH 7B0
```

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

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

The final setup statement that you will need is to tell the ELM329 to send messages to CAN Extended Address 04:

```
>AT CEA 04
```

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

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

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

That's all. To exactly reproduce the flow of data shown previously, you only need to send the relevant data bytes and the ELM329 will add the rest:

```
>10 81
50 81

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

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

CAN Input Frequency Matching

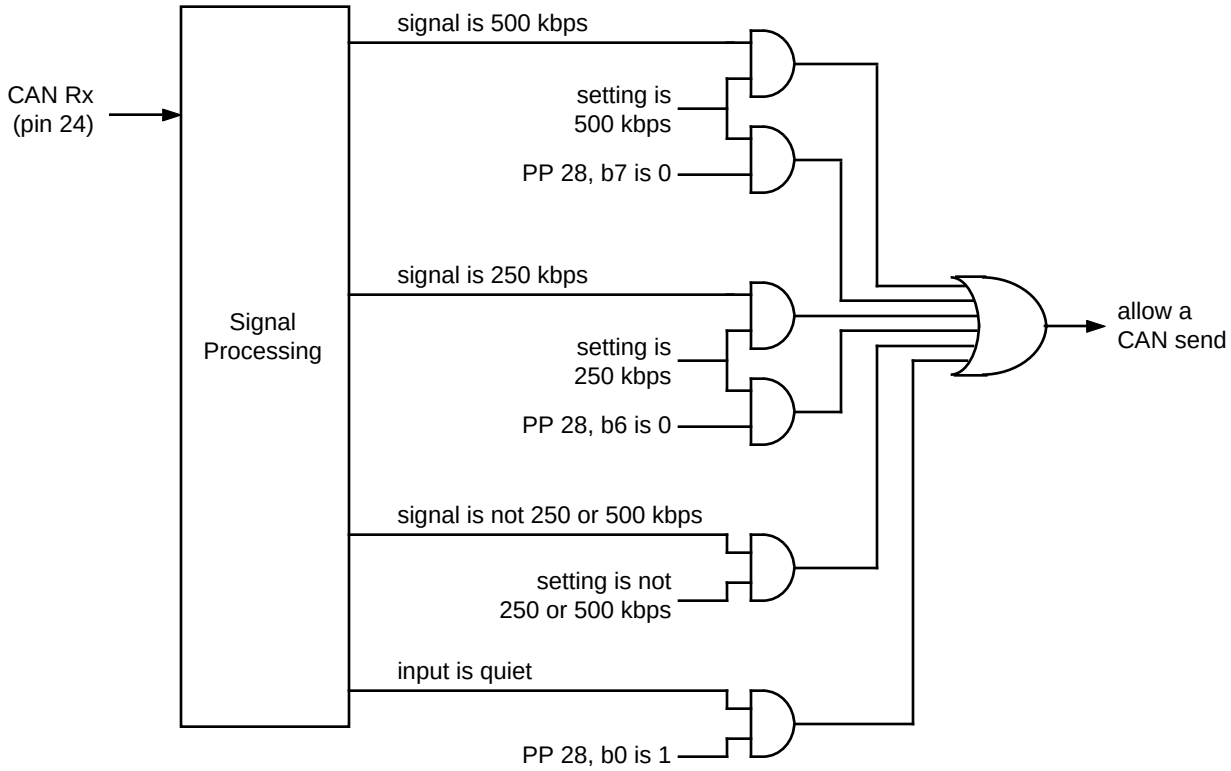
Most modern vehicles have a CAN network connected to pins 6 and 14 of the OBD connector. At one time, however, the use of these pins was left to the vehicle manufacturer, and a number of different systems were connected to them.

In order to prevent the disruption of any connected systems while the ELM329 is searching for a protocol (it sends out requests during a search), the ELM329 now performs several tests on these wires. Prior to firmware version 2.1, the tests simply looked for activity on the wires but were not frequency selective. This meant that, for example, vehicles that had a speedometer signal connected to either pin might be seen as a valid CAN network, and the ELM329 may have sent a request on these wires. The new firmware actually measures the input frequency and requires that it matches that of the selected CAN protocol before any test message can be sent.

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

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

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



Send Logic While Searching for a Protocol

CAN (Single Wire) Transceiver Modes

The ELM329 was designed with two wire CAN (OBDII) applications in mind, but there is no reason that it can not be used for single wire CAN applications (SAE J2411, etc.), as well. The data format remains much the same on the CAN networks - it is really the physical interface that differs.

Single wire CAN transceiver chips are available and should be used when connecting the ELM329 to single wire CAN networks. These ICs usually provide mode inputs which are used to change the state of the device - to put it into low power sleep mode, set the output to high voltage (12V) mode, etc. The table below shows the four modes typically supported by single wire CAN transceiver ICs, and the mode inputs most often used for each.

The ELM329 provides two output pins (M0 and M1) that may be used to set the modes for a single wire CAN transceiver. After every reset or AT D command, the level at pins 21 and 22 will be set according to PP 20. Note that firmware v1.0 set these pins to a low level (mode = sleep) when the IC went to low power mode, but the ELM329 no longer changes the setting while in low power mode.

The M0 and M1 pin levels are controlled with the Transceiver Mode commands. For example, if you wish to put the transceiver into the high voltage wakeup mode, simply send;

```
>AT TM 2
OK
```

and to restore the mode to normal, send:

```
>AT TM 3
OK
```

If you do not require these pins for a single wire CAN application, they may be used as general purpose outputs, much like the Control output.

TM #	M1 (pin 21)	M0 (pin 22)	Mode
0	0	0	Sleep
1	0	1	High Speed
2	1	0	High Voltage Wakeup
3	1	1	Normal

Single Wire Transceiver Modes

Programming Serial Numbers

A number of our customers have asked for ways to uniquely identify a product that uses our ELM329 integrated circuit. While this is often a request for a means to store a 'serial number', people have also asked for a way to store dates, and version codes, too. The @2 and @3 commands were created to assist with this.

If you send the command AT @2 to a new ELM329 integrated circuit, you will receive an error. That is, you will see a response that looks like this:

```
>AT @2
?
```

In the above dialog, the ELM329 is trying to tell you that nothing has been programmed into this storage location yet.

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

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

If the correct number of bytes were received, and programmed into internal memory, you see the 'OK' response. This 'number' can never be altered once it has been set, so you must be sure that you are entering the values properly. If developing code which does this, you may find that purchasing an ELM328 IC will save the expense of trial and error as it does allow reprogramming this storage location. The ELM328 IC does not support OBD protocols however, so is not a viable option for other uses.

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

```
>AT @2
MYBOARD_9906
```

... and may be used for chip identification.

Saving a Data Byte

The ELM329 provides a single memory location that can be used to save any one byte of information. This location uses special 'non-volatile' EEPROM memory for storage, so your data is not lost even if you should turn the power off.

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

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

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

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

```
>AT RD
7F
```

Since this single byte of data is stored in the internal EEPROM array, it is subject to the usual limits of EEPROM technology - unlimited reads, but typically only about 1 million writes, with a retention time of 40 years (or more). This should not be a problem for our ELM329 users, unless their software runs rampant.

The CAN Monitor (pin 11)

Version 2.0 of the ELM329 introduced a CAN monitor module that could be used to control the switching to and from the Low Power mode. The module uses an internal counter that is dedicated to continually monitoring the CAN receive signal, so it is able to catch even the shortest burst of CAN activity.

The CAN Monitor count is monitored by the firmware, and is typically tested every 20 msec to see if there has been any activity. The logic looks for eight or more counts (which would be about 400 Hz if the signal were continuous), before it considers the input to be active. Should activity stop, this can initiate the Low Power mode of operation, if PP 0E and PP 0F are set for this (see the Low Power Mode section for details).

The firmware also checks the CAN Monitor count regularly while in the Low Power mode. It looks for CAN activity to appear and may cause a switch to the normal (full power) mode, if allowed to do so. Permission to return to the normal mode is determined by bit 3 of PP 0F. If it is '1', then the ELM329 will 'wake up' as soon as the can signal appears. If it is '0', it will only wake up on the appearance of a CAN signal if there was a CAN signal present before it entered into the Low Power state.

Although this module is called a CAN Monitor, the signal presented to it does not have to be from a CAN system. If you prefer, you might connect something else to it, such as a tachometer or speedometer signal. As long as you restrict the voltage swing at the input pin, almost any signal can be connected. The CAN Monitor input (pin 11) has a very high input impedance, has Schmitt trigger waveshaping, advances the counter on the rising edge of the waveform, and can detect pulses as narrow as 30 nsec.

Note that versions 2.0 and 2.1 of the ELM329 tried to use pin 11 to also control the flashing of the Active LED while the ELM329 was in the Low Power mode. This was too confusing for many and has been changed in version 2.2 of the firmware. Pin 11 is now only used for the CAN Monitor, and the flashing Active LED is controlled by b4 of PP 0F.

Control Module Operation

The ELM329 provides two general purpose inputs and one general purpose output that you may use for your own control applications.

The two inputs are provided for monitoring signals that you connect. They both have Schmitt trigger wave-shaping on the input so can accommodate even the slowest moving signals. These inputs should also be protected from voltages which exceed the supply limits (usually a series resistance is all that is need for this).

Reading the level at an input is simply a matter of sending the appropriate AT command. For pin 12, send:

```
>AT IN1
0
```

and the ELM329 reports the logic level at the input (the '0' in this case). Similarly, the level at pin 13 is read with:

```
>AT IN2
1
```

The Control output (pin 4) may be set to a high or low level at any time with the AT C command. To set the Control output high, simply send:

```
>AT C1
```

and to set it low, send:

```
>AT C0
```

After a system reset (power on, MCLR, AT Z or AT WS), the Control output is always initialized to a low level.

Note that the Control output can also be selected to show the internal CAN activity signal (as determined by the CAN Monitor at pin 11). Simply set PP 0F bit 0 to '1' in order to enable it.

There are no restrictions on how you use these inputs and the output, as long as the electrical limits aren't exceeded. You may wish to control a buzzer, perhaps an LED, or to monitor a switch input or voltage level - it's up to you.

Low Power Mode

Often, the ELM329 is connected to a vehicle for only a short time, so power consumption is not of great concern. Occasionally, the ELM329 may be connected for longer times, however, possibly without the engine running. For these applications, it is often desirable to be able to put the circuit into a low power 'standby' state, and have it return to normal operation when needed. The power control features of the ELM329 are provided for this.

There are four ways in which the ELM329 can be placed into the low power standby mode (these are shown pictorially in Figure 6 below). They all require that the master enable bit (bit 7 of PP 0E) be set to '1' for them to function (which it is by default).

The first method is with an AT command. Simply send 'AT LP' at the prompt:

```
>AT LP
```

and the ELM329 will go to the low power mode after a one second delay (which provides a controlling circuit with a little time to perform some housekeeping tasks).

When the ELM329 goes to the low power mode, it first turns all five LED outputs off, then switches pin 14

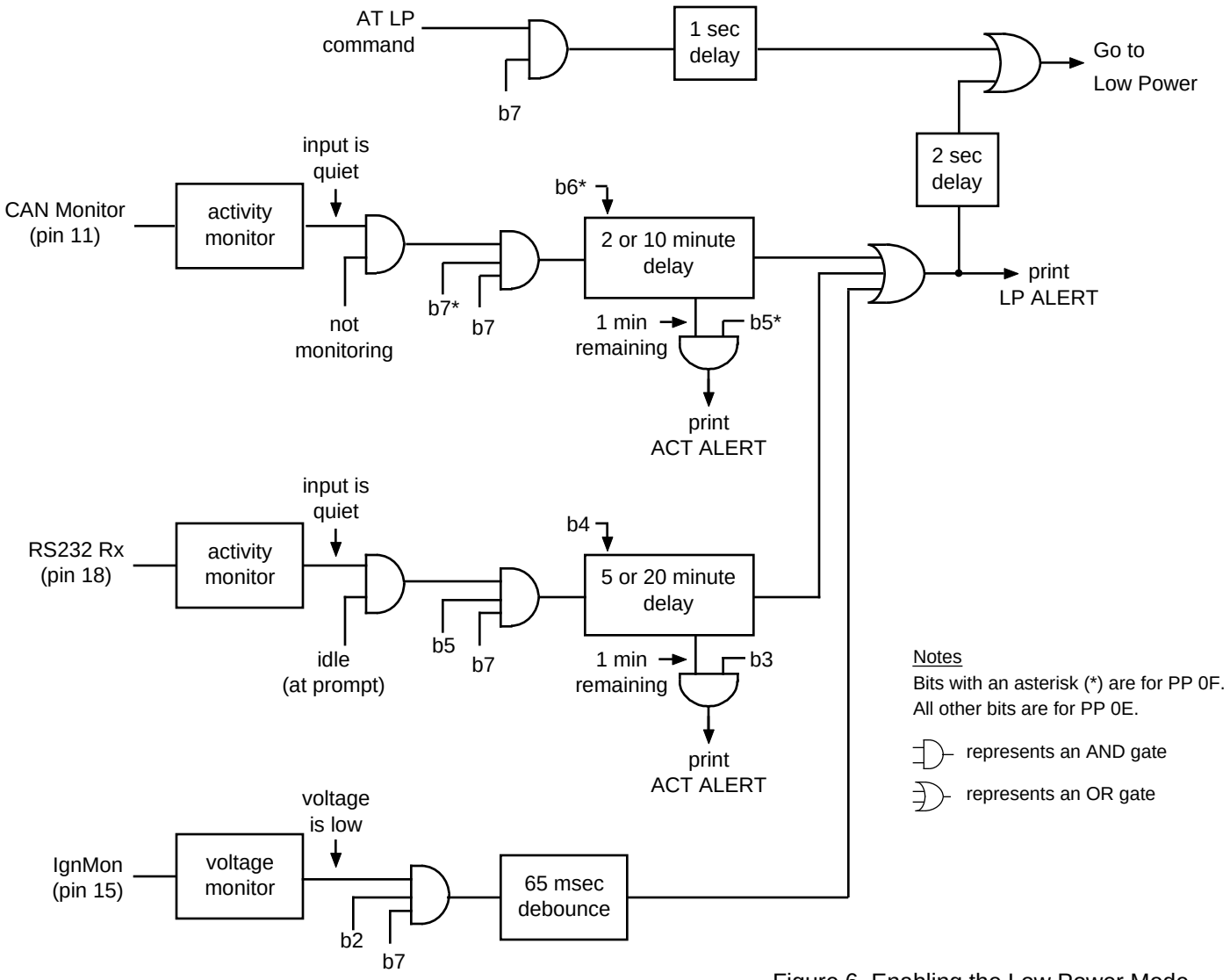


Figure 6. Enabling the Low Power Mode

Low Power Mode (continued)

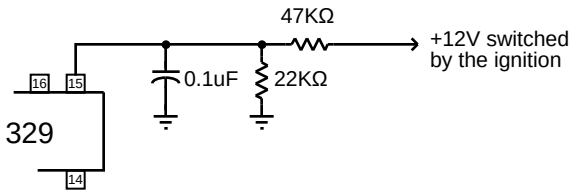
to its low power setting. After a delay of 50 msec (which gives the power supplies a chance to settle), pin 16 is then set to its low power level, and then after a further 50 msec, the chip reduces its power needs to a very low level. Note that while firmware version 1.0 also set M0, M1 and the Control output to a low level, that is not the case as of v2.0 - those 3 pins will now maintain their levels while in the low power mode.

The CAN Monitor (new with v2.0) offers another way in which the ELM329 can be switched to the low power mode. This module continually monitors pin 11 for a signal, and can initiate the low power mode if the signal disappears for more than either 2 or 10 minutes. To use this monitor, simply connect pin 11 to the CAN Rx pin.

Note that the CAN Monitor signal is not able to cause a switch to the low power mode if the IC is in a monitoring mode (AT MA, DM1 or MP). This logic is provided to aid in troubleshooting, as you likely do not want the circuit to suddenly power down while you are trying to capture some rare data. Also of interest is the fact that a CAN transceiver will receive its own sends, so if you are sending data on the CAN bus (perhaps periodic messages), this will be seen by the CAN Monitor as activity, and will prevent a switch to low power mode.

The third method is very similar in function to the CAN Monitor. It allows automatic switching to the low power mode when there has been no RS232 input for a period of time (which is useful if your controlling computer may be turned off at any time). This method does not require any wiring changes as the connection has been made internally.

The final method that may be used to enter the low power mode is by a low level appearing at the ignition monitor input (pin 15). It is enabled by setting both b2 and b7 of PP 0E to '1'. Note that when connecting to pin 15, care must be taken to not pass excessive current (>0.5 mA) through the internal protection diodes - a series resistor must be used to prevent this. Typically a circuit like this works well:



A relatively large capacitor (0.1 µF) was chosen for this circuit to provide filtering for noisy signals. This should not be a problem, as pin 15 uses a Schmitt trigger waveshaping circuit on the input.

In addition to the capacitor filtering, the IgnMon input logic uses a short internal delay ('debounce') timer to be sure that the low level is a legitimate 'key off', and not just noise spikes. As with the previous two methods, when the low power mode is initiated, the ELM329 will send an alert message ('LP ALERT'), wait 2 seconds, and will then begin low power mode.

The AT IGN command can always be used to read the level at pin 15, regardless of the setting of the PP 0E enable bits. This may be used to advantage if you wish to manually shut down the IC using your own timing and criteria. Recall that the alternate function for pin 15 is the RTS input which will interrupt any OBD processing that is in progress. So, if the ELM329 reports being interrupted (ie 'STOPPED'), you can then check the level at pin 15 with the AT IGN command, and make your own decisions as to what should be done. For that matter, you don't even need to reduce the power based on the input - you might possibly do something entirely different.

Having put the ELM329 into low power mode, you will need a means to wake it up. There are several ways in which you may do this (note that it does not have to be the same method that put it into the low power mode). Figure 7 is a block diagram which shows the possible ways.

The first way that a wakeup may occur is by the CAN Monitor sensing that there has been a change from no CAN activity to there being activity (this is shown as a 'rising edge' condition in the diagram). 'CAN activity' is defined as a series of pulses or the first edge of an active CAN bit with b2 of PP 0F (by default, a series of pulses are required). In addition to the appearance of CAN activity, you may also require that the CAN input was previously active by setting (PP 0F) bit 3 to 0 (it is 1 by default). This bit 3 switch was provided to more or less ensure that the circuit only wakes up on CAN activity if that was the cause of it going to low power mode. PP 0F bit 7 does not have to be set in order for the circuit to 'wake up' on CAN activity.

The other two ways that may be used to 'wake' the circuit are also shown in Figure 7. The first is with an RS232 input that goes to the active (low) level for at least 128 µsec. This may be accomplished by sending a space or @ character if the baud rate is less than

Low Power Mode (continued)

about 57.6 kbps. At higher baud rates, it may be more difficult to generate this width, so you might consider temporarily shifting to a lower baud rate, or see if your software can generate a 'break' signal. If you are directly connected to a microprocessor, then you might be able to generate this pulse output in software.

The final method to wake the ELM329 is with a low to high signal level transition at the IgnMon input. If this is wired (through resistors) to a 12V signal that is controlled by the ignition switch, then turning on the ignition will also turn on the ELM329 circuit.

No matter how the ELM329 is brought back to full power operation, it will always restore pin 14 first, followed 50 msec later by pin 16, and then it will wait 1 second before proceeding with the startup. This delay gives your external circuitry time to start up before being expected to be fully functional.

After the 1 second period, the ELM329 will then perform a partial warm start (AT WS) and be ready for operation. We say partial because several settings are not altered by the wakeup. The settings that remain unchanged are:

AT0/1/2	CAF0/1	CEA	CER*
CFC0/1	CSM0/1	D0/1	E0/1
H0/1	JTM1/5	L0/1	M0/1
R0/1	S0/1	STM1/5	

* only if CEA was active

In addition, the protocol number is not reset (but the protocol is closed).

This has discussed some of the aspects of using the Power Control feature, from a logical perspective. Refer to the 'Modifications for Low Power Standby Operation' section (on page 84) for some electrical design considerations.

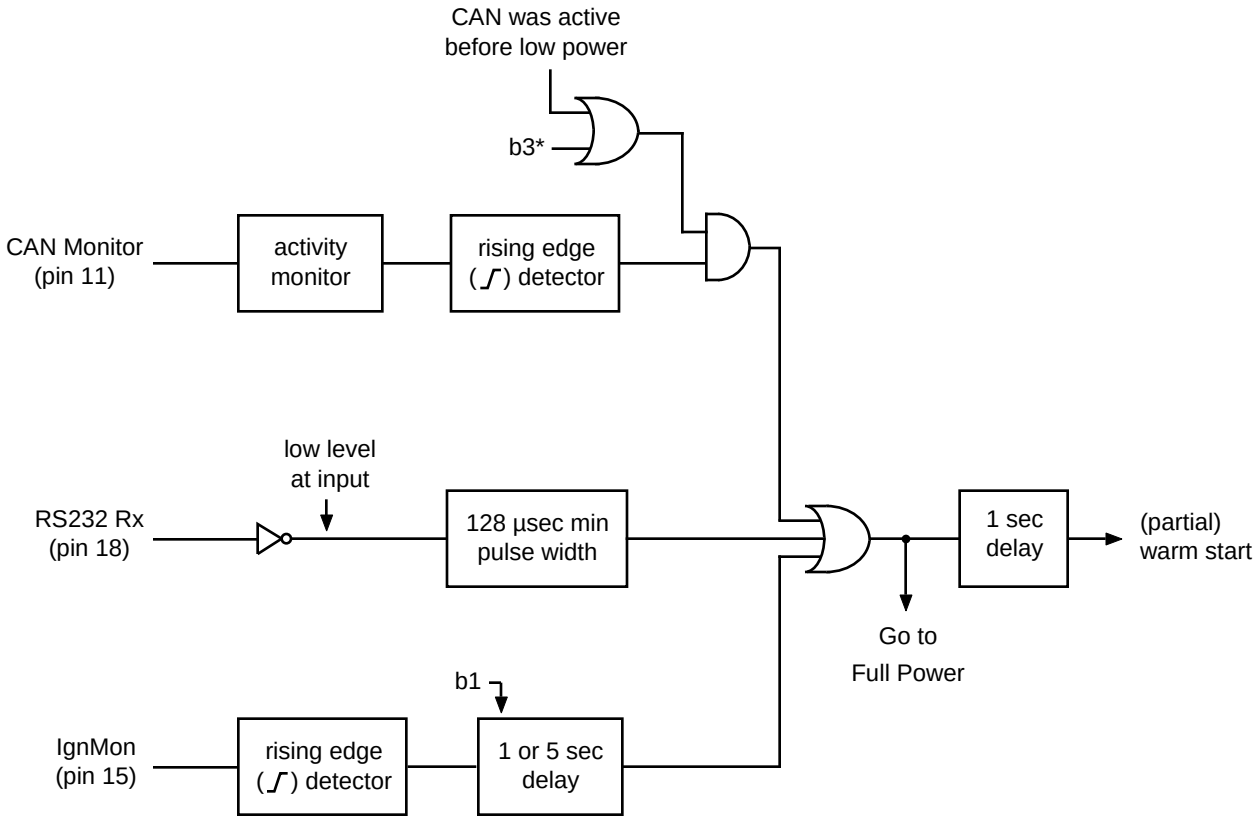


Figure 7. Returning to Normal Operation

Programmable Parameters

The ELM329 contains several programmable memory locations that retain their data even after power is turned off. Every time the IC is powered up, these locations are read and used to change the default settings for such things as whether to display the headers, or how often to send 'wakeup' messages.

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

The following pages list the currently supported Programmable Parameters for this version of the ELM329. As an example of how to use them, consider PP 01 (shown on page 70) which sets the default state for the AT H command. If you are constantly powering your ELM329 and then using AT H1 to turn the headers on, you may want to change the default setting, so that they are always on by default. To do this, simply set the value of PP 01 to 00:

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

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

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

At this point, you have changed the default setting for AT H1/H0, but you have not changed the actual value of the current AT H1/H0 setting. From the 'Type' column in the table on page 70, you can see that the change only becomes effective the next time that defaults are restored. This could be from a reset, a power off/on, or possibly an AT D command.

With time, it may be difficult to know what changes you have made to the Programmable Parameters. To help with that, the ELM329 provides a Programmable Parameter Summary (PPS) command. This simply prints a list of all of the PPs, their current value, and whether they are on/enabled (N), or off/disabled (F). For an ELM329 v2.2 IC, with only the headers enabled (as discussed above), the summary table would look like this:

```
>AT PPS
00:FF F  01:00 N  02:FF F  03:19 F
04:01 F  05:FF F  06:F1 F  07:09 F
08:FF F  09:00 F  0A:0A F  0B:FF F
0C:68 F  0D:0D F  0E:9A F  0F:F8 F
10:FF F  11:FF F  12:FF F  13:FF F
14:FF F  15:FF F  16:FF F  17:FF F
18:FF F  19:FF F  1A:FF F  1B:FF F
1C:FF F  1D:FF F  1E:FF F  1F:FF F
20:03 F  21:FF F  22:62 F  23:00 F
24:00 F  25:00 F  26:00 F  27:FF F
28:FF F  29:FF F  2A:0C F  2B:02 F
2C:E0 F  2D:04 F  2E:E0 F  2F:0A F
30:42 F  31:01 F  32:F0 F  33:06 F
34:E0 F  35:0F F  36:FF F  37:FF F
```

You can see that PP 01 now shows a value of 00, and it is enabled (oN), while the others are all off.

Another example shows how you might change the CAN filler byte. Some systems use 'AA' as the value to put into unused CAN bytes, while the ELM329 uses '00' by default. To change the ELM329's behaviour, simply change PP 26:

```
>AT PP 26 SV AA
OK

>AT PP 26 ON
OK
```

Again, PP 26 is of type 'D', so the above change will not actually take effect until the AT D command is issued, or the ELM329 is reset.

The Programmable Parameters are a great way to customize your ELM329 for your own use, but you should do so with caution if using commercial software. Most software expects an ELM329 to respond in certain ways to commands, and may be confused if the carriage return character has been redefined, or if the CAN response shows data length codes, for example. If you make changes, it might be best to make small changes and then see the effect of each, so that it is easier to retrace your steps and 'undo' what you have done. If you get in too deeply, don't forget the 'all off' command:

```
>AT PP FF OFF
```

No matter what software you use, you might get into more serious trouble, should you change the baud rate, or the Carriage Return character, for example, and forget what you have set them to. The Carriage Return value that is set by PP 0D is the only character that is recognized by the ELM329 as ending a

Programmable Parameters (continued)

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

When the ELM329 first powers up, it looks for a jumper between pin 28 (the OBD Tx LED output) and circuit common (V_{SS}). If a jumper is in place, it will turn off all of the PPs for you, restoring the IC to the factory defaults. To use this feature, simply connect a jumper to circuit common (which appears in numerous places - pins 8 or 19 of the ELM329, pin 5 of the RS232 connector, one end of most capacitors, or at the OBD connector), then hold the other end of the jumper to pin 28 while turning the power on. When you see the RS232 Rx LED begin to flash quickly, remove the jumper – the PPs are off.

This feature should only be used when you get into trouble too deeply, and it's your only choice (since putting a jumper into a live circuit might cause damage if you put it in the wrong place).

Programmable Parameter Summary

The following pages provide a list of the currently available Programmable Parameters. If a PP number is not shown in this table, then it does not currently have a function and it will not affect the operation of the ELM329.

All values shown are hexadecimal - the ELM329 does not recognize decimal numbers. Note that the 'Type' column indicates when any changes will take effect. The four possible values are as shown to the right (I, D, R or P).

I - the effect is Immediate,

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

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

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

PP	Description	Values	Default	Type
00	Perform AT MA immediately after powerup or reset	00 = ON FF = OFF	FF (OFF)	R
01	Printing of header bytes (AT H default setting)	00 = ON FF = OFF	FF (OFF)	D
03	NO DATA timeout time (AT ST default setting) setting = value x 4.096 msec	00 to FF	19 (102 msec)	D
04	Default Adaptive Timing mode (AT AT setting)	00 to 02	01	D
06	OBD Source (Tester) Address. Not used for J1939 protocols.	00 to FF	F1	R
07	Last Protocol to try during automatic searches	06 to 0F	09	I
08	Display a false ID ('ELM327 v2.2')	00 = ON FF = OFF	FF (OFF)	R
09	Character echo (AT E default setting)	00 = ON FF = OFF	00 (ON)	R

Programmable Parameter Summary (continued)

<table>
<tr><th>PP</th><th>Description</th><th>Values</th><th>Default</th><th>Type</th></tr>
<tr><td>0A</td><td>Linefeed Character</td><td>00 to 20</td><td>0A</td><td>R</td></tr>
<tr><td>0C</td><td>RS232 baud rate divisor when pin 6 is high (logic 1)
The baud rate (in kbps) = 4000 ÷ (PP 0C value)
For example, 500 kbps requires a setting of 08 (as 4000/8 = 500)
Here are some example baud rates, and the divisor to be used:
<table>
<tr><th>Baud Rate (kbps)</th><th>PP 0C value (hex)</th></tr>
<tr><td>19.2</td><td>D0</td></tr>
<tr><td>38.4</td><td>68</td></tr>
<tr><td>57.6</td><td>45</td></tr>
<tr><td>115.2</td><td>23</td></tr>
<tr><td>230.4</td><td>11</td></tr>
<tr><td>500</td><td>08</td></tr>
</table>
Notes:
1. The PP 0C value must be provided as hex digits.
2. The ELM329 can only process continuous byte receives at rates of 700 kbps or less. If you need to connect at a higher rate, consider adding a delay between the bytes to maintain an average rate of 70K bytes per second, or less.
3. A value of 00 provides a baud rate of 9600 bps.</td><td>00 to FF</td><td>68
(38.4)</td><td>P</td></tr>
<tr><td>0D</td><td>Carriage Return Character
used to detect and send line ends</td><td>00 to 20</td><td>0D</td><td>R</td></tr>
<tr><td>0E</td><td>Power Control options
Each bit of this byte controls an option, as follows:
b7: Master enable 0: off 1: on
if 0, pins 15 and 16 perform as RTS and Busy
(must be 1 to allow any low power functions)
b6: Pin 16 full power level 0: low 1: high
normal output level, is inverted when in low power mode
b5: Auto LP (RS232) control 0: disabled 1: enabled
allows low power mode if the RS232 activity stops
b4: Auto LP (RS232) timeout 0: 5 mins 1: 20 mins
no RS232 activity timeout setting
b3: Auto LP (RS232) warning 0: disabled 1: enabled
if enabled, says ‘ACT ALERT’ 1 minute before timeout</td><td>00 to FF</td><td>9A
(10011010)</td><td>R</td></tr>
</table>

Programmable Parameter Summary (continued)

PP	Description	Values	Default	Type
0E	Power Control options (continued) b2: Pin 15 function 0: RTS 1: IgnMon for use as IgnMon input, b7 must also be '1' b1: Ignition delay 0: 1 sec 1: 5 sec delay after IgnMon (pin 15) returns to a high level, before normal operation resumes b0: reserved for future - leave set at 0			
0F	More Power Control options Each bit of this byte controls an option, as follows: b7: Auto LP (CAN) control 0: disabled 1: enabled allows low power mode if the CAN activity stops b6: Auto LP (CAN) timeout 0: 2 mins 1: 10 mins no CAN activity timeout setting b5: Auto LP (CAN) warning 0: disabled 1: enabled if enabled, says 'ACT ALERT' 1 minute before timeout b4: Active LED 0: off 1: flashes setting controls flashing during low power b3: Previous CAN 0: required 1: ignored wakeup on CAN activity may be set to require that there was CAN activity prior to going to low power mode b2: CAN activity definition 0: pulses 1: low level setting to 1 allows fast response to 'bus wakeup' signals b1: reserved for future - leave set at 0 b0: Control output 0: normal 1: CAN Monitor if 1, then the pin 4 output follows CAN activity at pin 11 (i.e. is a high level when there is CAN activity detected)	00 to FF	F8 (11111000)	R
20	Default (Single Wire) Transceiver Mode M0 - M1 pin setting during normal CAN operation (see page 25)	00 to 03	03 (normal)	D
21	Default CAN Silent Monitoring setting (for AT CSM)	FF = ON 00 = OFF	FF (ON)	R
22	CAN wakeup message rate (AT SW default setting) setting = value x 20.48 msec	01 to FF	62 (2.0 sec)	D
23	Default Wakeup Mode (AT WM setting)	00 to 02	00 (OFF)	D
24	CAN auto formatting (AT CAF default setting)	00 = ON FF = OFF	00 (ON)	D
25	CAN auto flow control (AT CFC default setting)	00 = ON FF = OFF	00 (ON)	D

Programmable Parameter Summary (continued)

PP	Description	Values	Default	Type
26	CAN filler byte (used to pad out messages)	00 to FF	00	D
28	CAN Filter outputs (controls CAN sends while searching) The bits of this byte control options, as follows: b7: 500 kbps match 0: ignored 1: required b6: 250 kbps match 0: ignored 1: required b5 to b1: reserved for future - leave set to 1 b0: send if bus is quiet 0: not allowed 1: allowed	00 to FF	FF (11111111)	D
29	Printing of the CAN data length (DLC digit) when printing header bytes (AT D0/D1 default setting)	00 = ON FF = OFF	FF (OFF)	D
2A	CAN Error Checking (controls testing) Each bit of this byte controls an option, as follows: b7: ISO 15765 Data Length 0: accept any 1: must be 8 bytes Allows acceptance of non-standard data without errors. b6: ISO 15765 PCI=00 0: allowed 1: not allowed Some vehicles send 00's, which can be confusing. b5: reserved for future - leave set to 0 b4: reserved for future - leave set to 0 b3: Wiring Test 0: bypass 1: perform Certain wiring conditions may cause problems. This allows a quick test, which weeds out some problems. Processing 7F xx 78's: b2: enabled 0: no 1: yes b1: valid Modes (xx values) 0: all 1: only 00 to 0F b0: valid CAN protocols 0: all 1: only ISO 15765	00 to FF	0C (00001100)	D
2B	Protocol A (SAE J1939) CAN baud rate divisor. The baud rate is determined by the formula: rate (in kbps) = 500 ÷ value For example, setting this PP to 19 (ie. decimal 25) provides a baud rate of 500/25 = 20 kbps.	01 to 40	02 (250 Kbps)	R
2C	Protocol B (USER1) CAN options. Each bit of this byte controls an option, as follows: b7: Transmit ID Length 0: 29 bit ID 1: 11 bit ID b6: Data Length 0: fixed 8 byte 1: variable DLC b5: Receive ID Length 0: as set by b7 1: both 11 and 29 bit b4: baud rate multiplier 0: x1 1: x 8/7 (see note 2)	00 to FF	E0 (11100000)	R

Programmable Parameter Summary (continued)

PP	Description	Values	Default	Type
2C	Protocol B (USER1) CAN options (continued) b3: reserved for future - leave set at 0. b2, b1, and b0 determine the data formatting options: b2 b1 b0 Data Format 0 0 0 none 0 0 1 ISO 15765-4 0 1 0 SAE J1939 Other combinations are reserved for future updates – results will be unpredictable if you should select one of them.	00 to FF	E0 (11100000)	R
2D	Protocol B (USER1) baud rate divisor. See PP 2B for a description.	01 to 40	04 (125 Kbps)	R
2E	Protocol C (USER2) CAN options. See PP 2C for a description.	00 to FF	E0 (11100000)	R
2F	Protocol C (USER2) baud rate divisor. See PP 2B for a description.	01 to 40	0A (50 Kbps)	R
30	Protocol D (USER3) CAN options. See PP 2C for a description.	00 to FF	42 (01000010)	R
31	Protocol D (USER3) baud rate divisor. See PP 2B for a description.	01 to 40	01 (500 Kbps)	R
32	Protocol E (USER4) CAN options. See PP 2C for a description.	00 to FF	F0 (11110000)	R
33	Protocol E (USER4) baud rate divisor. See PP 2B for a description.	01 to 40	06 (95.2 Kbps)	R
34	Protocol F (USER5) CAN options. See PP 2C for a description.	00 to FF	E0 (11100000)	R
35	Protocol F (USER5) baud rate divisor. See PP 2B for a description.	01 to 40	0F (33.3 Kbps)	R
36	reserved for future use – leave set to FF	00 to FF	FF	–
37	reserved for future use – leave set to FF	00 to FF	FF	–

Notes:

1. For Programmable Parameter bits, b7 is the msb, and b0 is the lsb. For example, the PP 2C value of E0 is 11100000 in binary, in which b7 to b5 have the value ‘1’, while b4 to b0 have the value ‘0’.
2. When b4 of a CAN options PP is set, the CAN baud rate will be increased by a factor of 8/7. For example, if you set PP 2C b4 to ‘1’, and PP 2D to ‘06’, the actual baud rate will be 83.3 x 8/7 = 95.2 kbps. If you are unsure of your setting, the display protocol (AT DP) command may be used to display the actual baud rate.

Compatibility with the ELM327

In designing the ELM329, we have purposely maintained an almost identical pinout with our ELM327 integrated circuit. What this means is that you can remove the ELM327 chip from its circuit and simply insert an ELM329 chip, without causing damage to the ELM329.

There is only one consideration that we are aware of if you do this - the 510Ω resistors that are used for the 'K' and 'L' lines may get hot. This occurs because the default value for PP 20 is 03, which sets both the pin 21 and 22 outputs high. If you are going to use the ELM329 in an ELM327 circuit, we recommend that you either:

a) disconnect the 510Ω resistors (you may be able to simply lift one end), or

b) replace the 510Ω resistors with a higher value resistance (2K or greater), or

c) set PP 20 to 00

Any of the three will allow the ELM329 to function in an existing ELM327 circuit without generating excessive heat, or causing other problems.

Another consideration is the software, which is discussed in the next section.

Compatibility with ELM327 Software

The ELM329 uses many of the same instructions as the ELM327, and even pretends to support protocols 1 to 5 in order to be more compatible with it. This was part of the design of the ELM329, purposely done so that you might use ELM327 software with the ELM329.

While it seems that some ELM327 software does work well with the ELM329, some does not. Apparently some software checks to see if the IC identifies itself as an ELM327, and then refuses to work if it does not. This is quite frustrating for people that have purchased the software and simply want to compare results with the different ICs. To help those experimenters, we now include PP 08.

Programmable parameter 08 is used to tell the ELM329 to lie about its identity. If you set it to 00 as follows with:

```
>AT PP 08 SV 00
OK
```

and then enable the PP:

```
>AT PP 08 ON
OK
```

then follow this with a reset of the IC:

```
>AT Z
```

the ELM329 will then identify itself as an ELM327:

```
>AT I
ELM327 v2.2
```

Since the ELM329 does not exactly support all of the ELM327 commands, there may potentially be problems when using the ELM327 software with an ELM329. You will simply need to experiment to see if it works as you need with your existing software. Perhaps if it does work well, you might write to the author of the software telling them that it does, and suggesting that a future update might possibly accept either an ELM327 or an ELM329 ID string.

Maximum CAN Data Rates

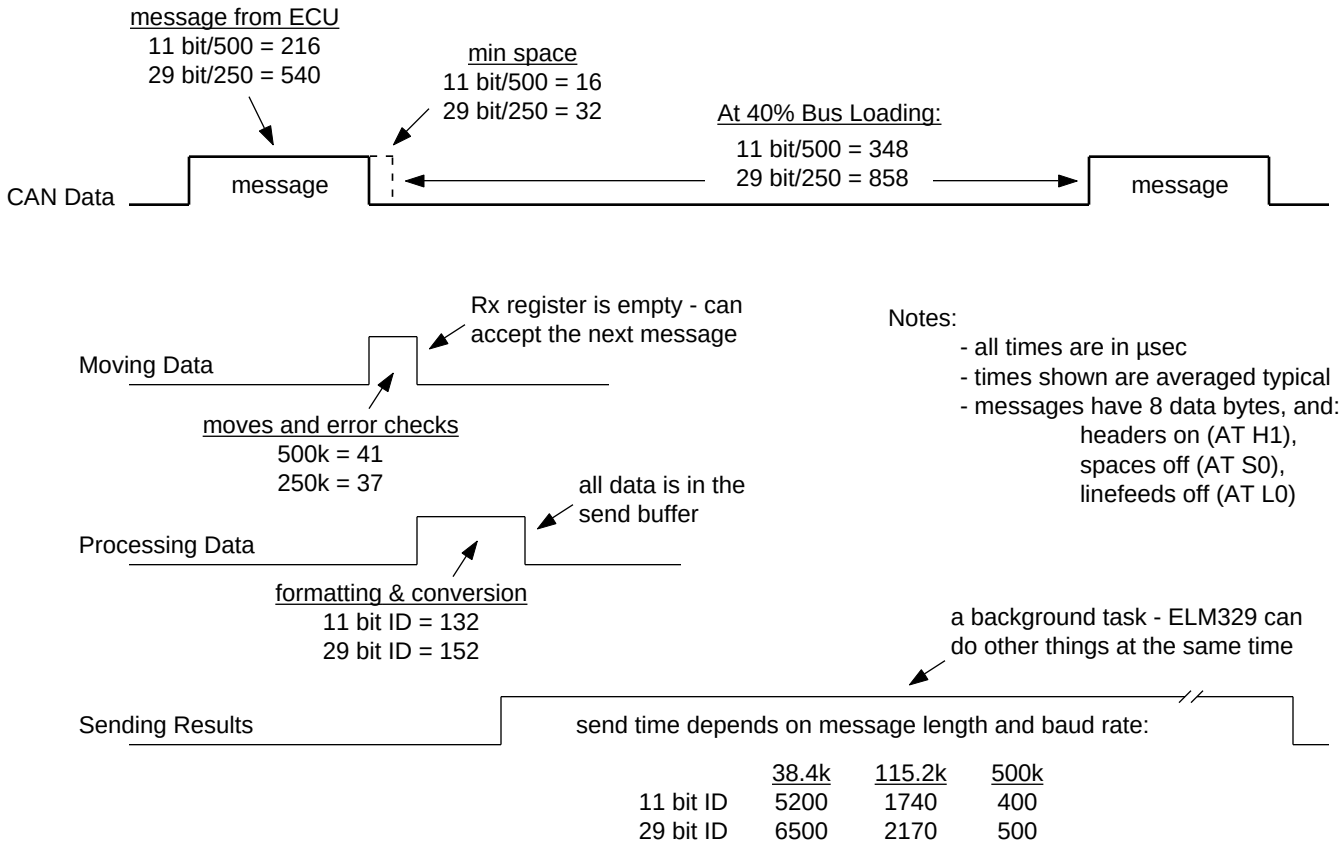
We are occasionally asked what the maximum data rate is that the ELM329 can handle. This is often after someone has tried to monitor all data using the default settings and has received a 'BUFFER FULL' error. It is difficult to say exactly what the maximum rate is, however, as several factors are involved.

The CAN 'engine' inside the ELM329 is actually configured with one receive register that accepts messages from the data bus, and another that accepts messages from the first. As long as the firmware empties the second register before the first register needs it, there should not be any overflow problems with this component. The ELM329 actually moves the data very quickly to temporary storage, so this is never a problem.

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

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

When a message arrives, the ELM329 moves quickly to move the received bytes from the special CAN registers, so that they do not affect the next message that arrives. The data is then formatted (as ASCII bytes) and placed into the RS232 transmit buffer, for sending to the controlling processor. As long as CAN messages do not arrive at a rate that is faster than the ELM329 can process them in, all messages



Maximum CAN Data Rates (continued)

will be processed. You can see from the figure that even for a 500 kbps message with an 29 bit ID, the ELM329 finishes the processing with time to spare. Since ISO15765-4 specifies that messages must be 8 data bytes in length (filler bytes are added as needed) these times do represent a typical situation with a 40% bus load. Actually, the ELM329 is able to process CAN messages in less time than it takes to transmit them, so it would be able to handle 100% bus loading (which is not a practical situation).

Once the ELM329 has placed all of the properly formatted bytes into the RS232 transmit buffer, it is up to the controlling computer to fetch them in a timely fashion. If the bytes are removed too slowly, the buffer will continue to fill as new OBD messages arrive, and it will eventually become full. It does not matter how big the buffer is, if the rate of removing bytes from the buffer is slower than the rate of putting them into the buffer, it will eventually fill up. When it is full, you will see the dreaded 'BUFFER FULL' message, and you will have to start over.

The ELM329 transmit buffer is 512 bytes in size. Considering that some bytes will be sent while new messages are being stored, this means that (with H1, L0 and S0 settings and 40% bus loading), you can typically save:

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

messages in the buffer. This storage is more than enough for OBD requests – the only time that you might get into trouble is if you are monitoring all messages on the bus (i.e. using AT MA) with no filters set. In that case, you would need to be sure that you are removing bytes as fast as they are being generated. Note that there are no numbers shown for 500 kbps as the buffer will not fill up when bus loading is at 40%.

Are you wondering what bus loading is? If you were to monitor a CAN data bus for a set period of time, and determine how much of that time was actually taken by CAN messages, the ratio of the two would be the 'bus loading'. This is a utilization factor that is very similar to the duty cycle for a square wave signal (and in the same way, it is normally expressed as a %). Ideally, bus loading should be less than about 30%, but as vehicles become more complex, this is very difficult to do. Some vehicles are reportedly seeing 70% bus loads.

When people ask us then, 'What data rate can the ELM329 support?' the answer is not easy to give, and depends on many factors. Certainly, there will be situations in which the IC is not able to get the data out as fast as you'd like, but that is often due to serial port limitations (whether through a poor choice of baud rates, or from hardware limitations). If you are simply obtaining standard OBD information, there is really no need to choose a high baud rate, as the 'buffer' takes care of temporary data storage for you. If you are trying to 'push the envelope', then hopefully this discussion has helped to give you the necessary background information.

Microprocessor Interfaces

A very common question that we receive is ‘Can I connect the ELM329 directly to my own circuit, or must I use the RS232 interface shown?’ Certainly you may connect directly to our ICs, and you do not need to use an RS232 interface. There are a few items to consider, however.

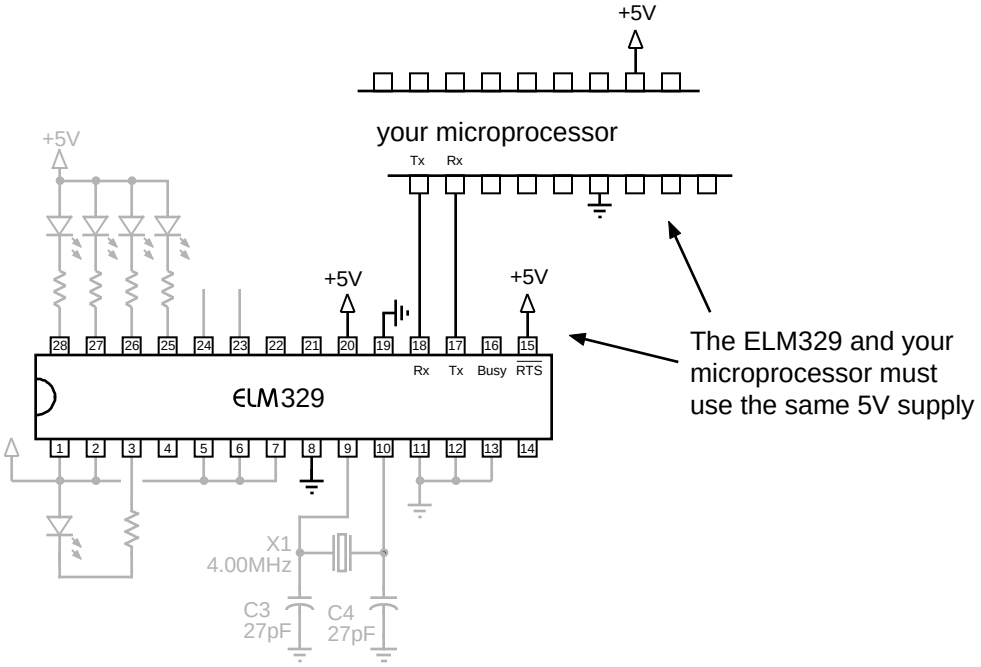
The ELM329 is actually a microprocessor that contains a standard UART type interface, connected to the RS232 Tx and Rx pins. The logic type is CMOS, and this is compatible with virtually all 5V TTL and CMOS circuits, so you should be able to connect directly to these pins provided that the two devices share the same power supply (5V), and that they are not physically more than about 10 to 20 inches apart (CMOS circuits are subject to latchup from induced currents, which may be a problem if you have long leads).

The normal (idle) levels of the ELM329 transmit and receive pins are at the V_{DD} (5V) level. Most microprocessors and RS232 interface ICs expect that to be the idle level, but you should verify it for your microprocessor before connecting to the ELM329. The connections are straightforward - transmit connects to receive, and receive connects to transmit, as shown below. Don’t forget to set both devices to the same baud rate.

The ELM329 also provides a hand-shaking feature that may simplify the flow of data for you (if you turn the low power feature off). The interface consists of two pins - an input an an output. The input is called ‘request to send’ (RTS), and it is used to interrupt the ELM329, just the same as tapping a key on the keyboard when using a terminal program. The output pin (‘Busy’) is used by the ELM329 to tell your system that it is processing data.

To use the handshaking feature, set one of your port pins to normally provide a high output, and connect it to the RTS input (pin 15). Use another port pin as an input to monitor the ELM329 Busy output (pin 16). When you want to send a command, simply check the Busy output first. If it is at a high logic level, then either wait for it to go low, or if you need to interrupt the IC, then bring the RTS line low and wait for the Busy line to go low. (You might want to consider using an edge triggered interrupt on the Busy output, if one is available). When Busy does go low, restore your RTS line to a high level, and then send your command to the ELM329. No need to worry about the ELM329 becoming busy again after you raise the RTS line at this point – once Busy goes low, the ELM329 will wait (indefinitely) for your command.

If you do not use the RTS input on the ELM329, it must be connected to a high logic level, as shown below.



Example Applications

The SAE J1962 (ISO 15031-3) standard dictates that all OBD compliant vehicles must provide a standard connector near the driver's seat, the shape and pinout of which is shown in Figure 8 below. The circuitry described here can be used to connect to this J1962 plug without modification to your vehicle.

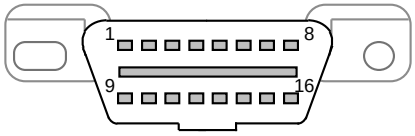


Figure 8. The J1962 Vehicle Connector

The male J1962 connector required to mate with a vehicle's connector may be difficult to obtain in some locations, and you might be tempted to improvise by making your own connections to the back of your vehicle's connector. If doing so, we recommend that you do nothing that would compromise the integrity of your vehicle's OBD network. The use of any connector which could easily short pins (such as an RJ11 type telephone connector) is definitely not recommended.

The circuit on page 80 (Figure 9) shows how the ELM329 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 need to use pin 4 instead of pin 5.) The regulator powers several points in the circuit as well as an LED (L6) for visual confirmation that power is present. We have shown an LP2950 for the regulator as that type has very low quiescent current which is important if you are going to use the low power feature of the ELM329.

Note that there are some electrolytic capacitors (C2 and C4) shown on the input and the output of the regulator. In addition to providing transient capability, the output one (C4) is also needed for regulator stability. Most types will work fine here, but solid tantalum or multi-layer ceramic are preferred.

The top left corner of Figure 9 shows the CAN interface circuitry. We do not advise making your own interface using discrete components – CAN buses may have a lot of critical information on them, and you can easily do more harm than good if you fail. It is strongly recommended that you use a commercial transceiver chip as shown. The Microchip MCP2561 is used in our circuit, but most major manufacturers also produce CAN transceiver ICs – look at the Microchip MCP2551, the NXP PCA82C251, the Texas Instruments SN65LBC031, the Infineon TLE7250G, and the Linear Technology LT1796, to name only a few. Be sure to pay attention to the voltage limits – depending on the application you may have to tolerate 24V, not just 12V. Also, note that some transceivers should have a resistor in series with pin 8, for the slew rate (slope) limiting function.

The voltage monitoring circuitry for the AT RV command is shown connected to pin 2 of the ELM329. The two resistors (R9 & R10) simply divide the battery voltage to a safe level for the ELM329, and the capacitor (C10) filters out noise. As shipped, the ELM329 expects a resistor divider ratio as shown, and sets nominal calibration constants assuming that. If your application needs a different range of values, simply choose the resistor values to maintain the input within the specified 0-5V limit, and then perform an AT CV to calibrate the ELM329 to your new divider ratio. The maximum voltage that the ELM329 can show is 99.9V.

The USB interface shown is a relatively new module from the people at Future Technology Devices International (FTDI - http://www.ftdichip.com). It has a 9 pin D-Sub footprint so is easily dropped into existing designs, and only needs a driver installation to be complete. Drivers are available for download from the FTDI web site - choose VCP Drivers and install the appropriate version for your PC. Note that in Figure 9, we show a diode (D3) and resistor (R11) installed between the FTDI interface and the ELM329. This is to prevent backfeeds from the USB supply, should the ELM329 circuit not be connected to a vehicle supply.

Since the default baud rate for the ELM329 is 38.4 kbps, you will need to set your COM port to that when first connecting. Once connected, you can change PP 0C to provide a different baud rate (but make sure you then change your PC's COM port settings as well).

The only remaining components are the LEDs and the crystal. The LEDs are standard ones, and may be any colour that you require - we only offer suggestions here. The crystal is a 4.000MHz microprocessor type, while the 27pF loading capacitors shown are typical only, (you may have to select other values depending on what is specified for the crystal that you use). This crystal frequency is critical to the circuit operation and must not be altered. Do not substitute a resonator for the crystal, as it will not have the accuracy required.

Example Applications (continued)

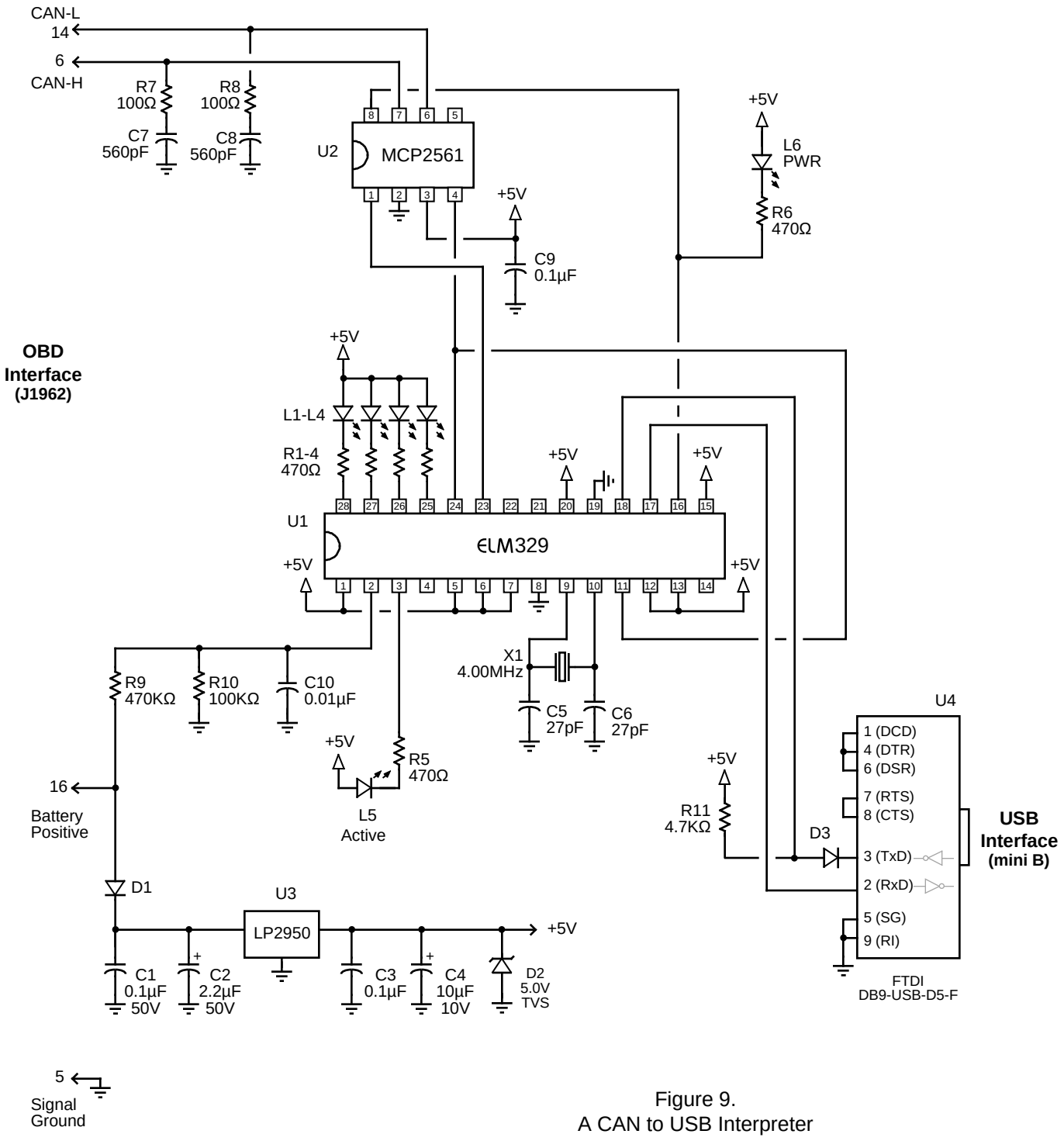


Figure 9.
A CAN to USB Interpreter

Example Applications (continued)

As always, you are not limited to the circuit of Figure 9. It is only a starting point that you can build on. The following pages show some alternative communications interfaces that may be of interest if you are considering modifying the circuit. Also, be sure to read our application note AN04 – ELM327 and Bluetooth on the web site under 'help' for a discussion on connecting with Bluetooth (the ELM327 information applies to the ELM329 as well).

Figure 11 shows a very basic RS232 interface that may be connected directly to pins 17 and 18 of the ELM329. 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. This circuit is limited to data rates of about 57.6 kbps, but has the advantage that it uses commonly available components and does not require a special integrated circuit.

The circuit of Figure 12 is useful if you wish to make a high speed RS232 interface. It uses a MAX3222E integrated circuit that only needs a few capacitors in order to function. These capacitors are used for an internal charge pump function that creates the dual polarity voltage supply that an RS232 output requires. One advantage to using the MAX3222E is that it can be put into a Low Power mode by the ELM329 (disconnect the +5V from pin 18, and then connect pin 18 to pin 14 of the ELM329).

If you use the MAX3222E, you must also modify the power supply of Figure 9. C2 should be increased to 10 µF, and C4 to 33 µF. Without these larger capacitors, you would likely experience 'LV RESET's as the MAX3222E transceiver is switched on and off. Testing has shown that these values for C2 and C4 should be adequate, but if you still see an occasional 'LV RESET', you may want to increase them even further.

The MAX3222E RS232 transceiver handles all of the voltage translation and logic inversion needed to provide standard RS232 communications between the ELM329 and the controlling computer. The 'E' version shown is capable of up to 250 kbps communications, which is likely enough for most applications. (The versions without the 'E' are only rated to 120 Kbps.)

In all, the MAX3222 has been a great chip, but we do caution you about one problem that we have encountered. If pin 18 (the Shutdown pin) is held low during powerup, then the chip will draw enough current

Semiconductors
D1 = 1N4001
D2 = 1N5232B or SA5.0AG TVS
D3 = 1N914 or 1N4148
L1, L2, L3, L4 = Yellow LED
L5 = Red LED
L6 = Green LED
U1 = ELM329 (CAN Interpreter)
U2 = MCP2561 (CAN Transceiver)
U3 = LP2950 (5V, 100mA regulator)
U4 = FTDI DB9-USB-D5-F
(UART Converter)

Misc
X1 = 4.000MHz crystal
ELM329 Socket = 28pin 0.3" (or 2 x 14pin)

Resistors (1/8W or greater)
R1 to R6 = 470 Ω
R7, R8 = 100 Ω
R9 = 470 KΩ
R10 = 100 KΩ
R11 = 4.7 KΩ

Capacitors (16V or greater, except as noted)
C1 = 0.1uF 50V
C2 = 2.2µF 50V
C3, C9 = 0.1uF
C4 = 10µF 10V
C5, C6 = 27pF
C7, C8 = 560pF 50V
C10 = 0.01µF

Figure 10. Parts List for Figure 9

Example Applications (continued)

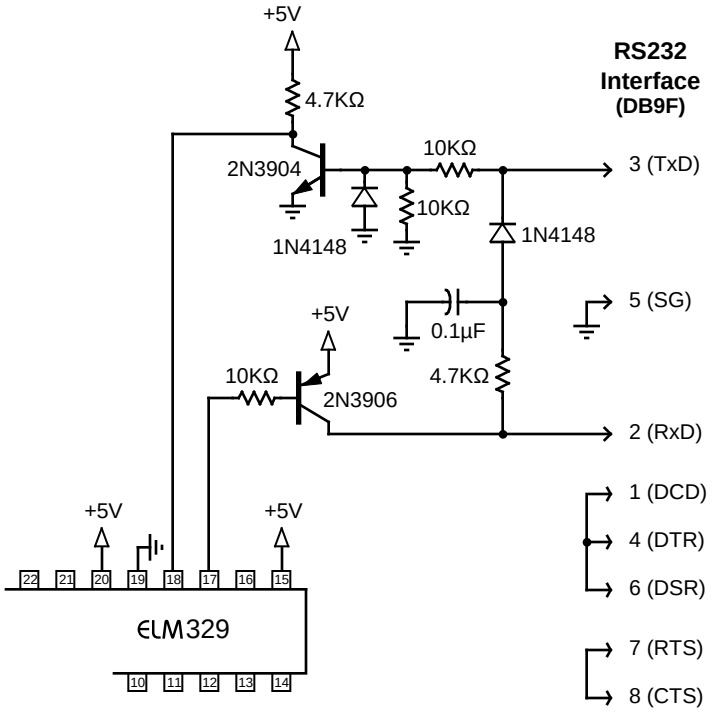


Figure 11. A Low Speed RS232 Interface (≤57.6 kbps)

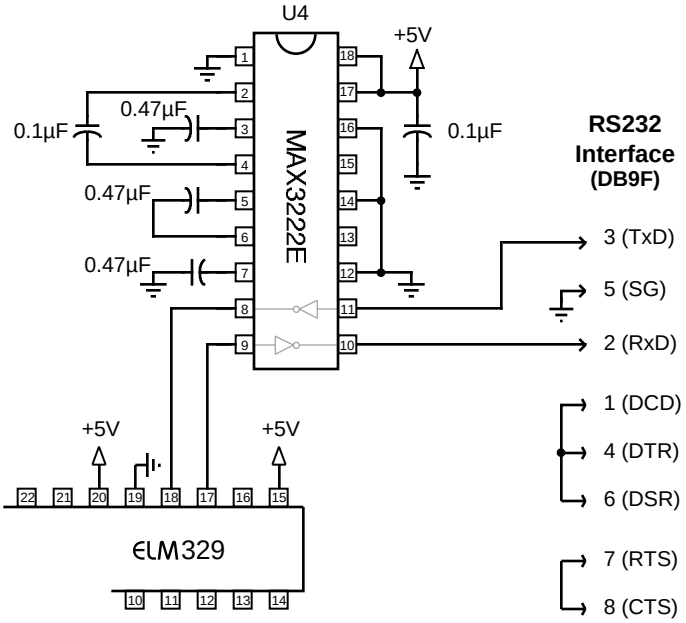


Figure 12. A High Speed RS232 Interface (≤250 kbps)

Example Applications (continued)

to drop the 5V supply to such a level that the ELM329 is not able to start up. Do not install any pull-down resistors on pin 18 of the MAX3222! If you build the circuit of Figure 12, you should not encounter any problems.

The final circuit (Figure 13) shows how you might connect the ELM329 to a different USB converter. While there are a few single IC products on the market that allow you to connect an RS232 system directly to USB, we show the CP2102 by Silicon Laboratories (www.silabs.com) here. It has proven to be fairly easy to connect and use. These ICs provide a very simple and relatively inexpensive way to 'bridge' between RS232 and USB, and as you can see, require very few components to support them.

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

This has provided some examples of how the ELM329 might be used in a circuit. Hopefully it has been enough to get you started on your way to many more.

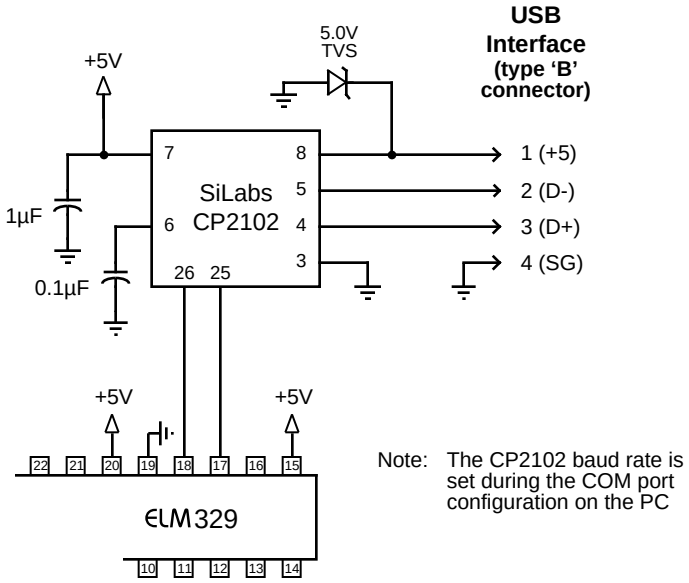


Figure 13. An Alternative USB Interface

Modifications for Low Power Standby Operation

The ELM329 may be placed in a low power standby mode in which it consumes very little current. This will find its greatest use with semi-permanent vehicle installations where you want the current consumption to be as low as possible (ideally zero) when the ELM329 is not needed.

Just how effective the low power mode is depends on your attention to detail when designing your circuit. If you use our example circuit of Figure 9, you will likely find that with 12.0V applied as 'Battery Positive', the measured current is typically:

base current (on the bench) = 25 mA

That is with the circuit powered on, but with no PC or ECU connected. If you connect it to a vehicle and a computer, the current typically rises to:

base current (in the vehicle) = 32 mA

with the Active LED on. When performing a constant function, such as 'monitoring all' data, this current rises further, and has been measured at:

active current (in the vehicle) = 46 mA

Any power supply designs should be able to supply this last current continuously, and be able to supply more than that under transient conditions (when transmitting, the data bus capacitance needs to be continually charged and discharged).

The "Low Power Mode" of operation section (page 66) discussed the ways in which you might initiate low power operation, but the easiest is to use the low power command (AT LP). After sending this, the total Figure 9 circuit current is typically:

current after AT LP = 0.25 mA

That is the typical current that you might expect with the example circuit. If you were to use the MCP2551 CAN transceiver instead of the MCP2561 as shown, the measured current would be slightly higher (typically about 0.50 mA). Note that whether the Active LED is set to flash or not has very little influence on this current (as it uses an average of about 25 µA). Similarly, the CAN Monitor typically only uses about 20 µA during low power operation, so does not appreciably affect the total current. If every µA counts in your design, they may be significant, however, and should be considered.

For typical vehicle applications, 0.25mA is a very low current, and is likely suitable for most applications. It would be difficult to reduce it further than that, but you may be able to do so, perhaps by eliminating the voltage monitoring circuit (R9 and R10), or by choosing a lower current voltage regulator. We leave those improvements to you.

Error Messages and Alerts

The following shows what the ELM329 will send to warn you of a condition or a problem. Some of these messages do not appear if using the automatic search for a protocol, or if the Programmable Parameter bits disable them.

ACT ALERT

This message occurs as a warning that there has been no CAN or RS232 activity (ie commands or messages received) for some time, and the IC is about to go to the low power mode in one minute. If an input signal is found during that minute, the switch to low power will be cancelled.

This message may be disabled by setting PP 0E bit 3 (RS232) or PP 0F bit 5 (CAN) to '0'.

BUFFER FULL

The ELM329 provides a 512 byte internal RS232 transmit buffer so that OBD messages can be received quickly, stored, and sent to the computer at a more constant rate. Occasionally (particularly with some CAN systems) the buffer will fill at a faster rate than it is being emptied by the PC. Eventually it may become full, and no more data can be stored (it is lost).

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

CAN ERROR

The CAN system had difficulty initializing, sending, or receiving. Often this is simply from not being connected to a CAN system when you attempt to send a message, but it may be because you have set the system to an incorrect protocol, or to a baud rate that does not match the actual data rate. It is possible that a CAN ERROR might also be the result of a wiring problem, so if this is the first time using your ELM329 circuit, review all of your CAN interface circuitry before proceeding.

If you are seeing these messages while working with the CAN silent mode off, then return to CAN silent mode (AT CSM1). There is likely a problem with your baud rate.

DATA ERROR

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

<DATA ERROR

There was an error in the line that this points to, either from an incorrect checksum, or a problem with the format of the message (the ELM329 still shows you what it received). There could have been a noise burst which interfered, possibly a circuit problem, or perhaps you have the CAN Auto Formatting (CAF) on and you are looking at a system that is not of the ISO 15765-4 format. Try resending the command again – if it was a noise burst, it may be received correctly the second time.

ERRxx

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

One error that is not necessarily a result of an internal problem is ERR94. This code represents a 'fatal CAN error', and may be seen if there are CAN network issues (some non-CAN vehicles may use pins 6 and 14 of the connector for other functions, and this may cause problems). If you see an ERR94, it means that the CAN module was not able to reset itself, and needed a complete IC reset to do so. You will need to restore any settings that you had previously made, as they will have returned to their default values.

LP ALERT

This appears as a warning that the ELM329 is about to switch to the low power (standby) mode of operation in 2 seconds time. This delay is provided to allow an external controller enough time to prepare for the change in state. No inputs or voltages on pins can stop this action once initiated.

Error Messages and Alerts (continued)

LV RESET

The ELM329 continually monitors the 5V supply to ensure that it is within acceptable limits. If the voltage should go below the low limit, a 'brownout reset' circuit is activated, and the IC stops all activity. When the voltage returns to normal, the ELM329 performs a full reset, and then prints LV RESET. Note that this type of reset is exactly the same as an AT Z or MCLR reset (but it prints LV RESET instead of ELM329 v2.2).

This low voltage protection is not only necessary for the ELM329 to operate properly, but it also offers protection for the CAN transceiver ICs too. Most transceiver chips require a minimum operating voltage of 4.5V, while some require a minimum of 4.75V.

NO DATA

The IC waited for the period of time that was set by AT ST, and detected no response from the vehicle. It may be that the vehicle had no data to offer for that particular PID, that the mode requested was not supported, that the vehicle was attending to higher priority issues, or possibly that the filter was set so that the response was ignored, even though one was sent. If you are certain that there should have been a response, try increasing the ST time (to be sure that you have allowed enough time for the ECU to respond), or restoring the CAN filter to its default setting.

<RX ERROR

An error was detected in the received CAN data. This most often occurs if monitoring a CAN bus using an incorrect baud rate setting, but it may occur if monitoring and there are messages found that are not being acknowledged, or that contain bit errors. The entire message will be displayed as it was received (if you have filters set, the received message may not agree with the filter setting). Try a different protocol, or a different baud rate.

SEARCHING

This will be displayed when a message has been provided for transmitting, but a protocol is not yet considered to be active. When displayed, it means that the ELM329 is searching for an appropriate protocol to use. Similarly, the message may be displayed if the IC is directed to monitor the data bus, and there is no protocol active.

STOPPED

If any OBD operation is interrupted by a received RS232 character, or by a low level on the RTS pin, the ELM329 will print the word STOPPED. If you should see this response, then something that you have done has interrupted the ELM329. Note that short duration pulses on pin 15 may cause the STOPPED message to be displayed, but may not be of sufficient duration to cause a switch to low power operation.

The message is printed any time that an OBD task is interrupted.

UNABLE TO CONNECT

The ELM329 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, but it can also occur if you try to do something that is not appropriate for the protocol.

Version History

In many ways, the ELM329 is similar to our popular ELM327 IC, but it does offer several extra features. The following summarizes some of these for you:

v1.0

First version available commercially.

Features:

- supports ELM327 CAN protocols 6 to F, and allows protocol settings of 1 to 5.
- a CAN Monitor input
- Single Wire CAN transceiver controls
- an Active LED output.
- general purpose inputs and a Control output
- allows sending either 11 or 29 bit IDs regardless of protocol setting
- able to send CAN periodic ('keep-alive' or 'wakeup') messages.

AT Commands supported:

., :, <CR>, AT0, AT1, AT2, BD, BI, BRD, BRT, C0, C1, CAF0, CAF1, CEA, CF, CFC0, CFC1, CM, CP, CRA, CS, CSM0, CSM1, CV 0000, CV, D, D0, D1, DM1, DP, DPN, E0, E1, FCSD, FCSM, FCSH, H0, H1, I, IGN, IN1, IN2, JE, JHF0, JHF1, JS, JTM1, JTM5, L0, L1, LP, M0, M1, MA, MP, PB, PC, PP xx OFF, PP FF OFF, PP xx ON, PP FF ON, PP xx SV, PPS, R0, R1, RD, RTR, RV, S0, S1, SD, SH, SP, ST, SW, SW 00, TA, TM0, TM1, TM2, TP, V0, V1, WD, WH, WM, WM0, WM1, WM2, WS, Z, @1, @2, and @3

Programmable Parameters supported:

00, 01, 03, 04, 06, 07, 09, 0A, 0C, 0D, 0E, 20, 21, 22, 23, 24, 25, 26, 29, 2A, 2B, 2C, 2D, 2E, 2F, 30, 31, 32, 33, 34, and 35

v2.0

Features:

- RS232 transmit buffer increased to 512 bytes
- J1939 monitoring now displays Ack/Nack (E8) messages
- wake from Low Power mode now restores the AT0/1/2, CAF0/1, CEA, CFC0/1, CSM0/1, D0/1, E0/1, H0/1, JTM1/5, L0/1, M0/1, R0/1,S0/1, and TM0/1/2/3 settings.
- the Control output is able to follow CAN activity while in Low Power mode.

AT Commands:

- CA

Programmable Parameters:

- 0F

v2.1

Features:

- all automatic protocol searches now measure the CAN frequency if the bus is live. Sending is blocked if the actual frequency does not match the selected frequency
- brownout reset voltage setting reduced from 4.3V to 2.8V
- code changes for improved speed and reliability

AT Commands:

- no changes

Programmable Parameters:

- 28
- PP 07 default value changed to 9

v2.2

Features:

- detects Response Pending messages and waits for a response
- pin 11 functions only as a CAN Monitor
- wake from Low Power restores the STM1/5, and CER settings.

AT Commands:

- FB, STM1/5, IA, CER, W0/1, and WT
- added frequency display to CS
- removed the WM instruction

Programmable Parameters:

- 08
- PP 0C now allows 00 (9600 baud)
- PP 0A and 0D values must now be <$21
- PP 22 not allowed to be 00
- new PP 0F and PP 2A bit definitions

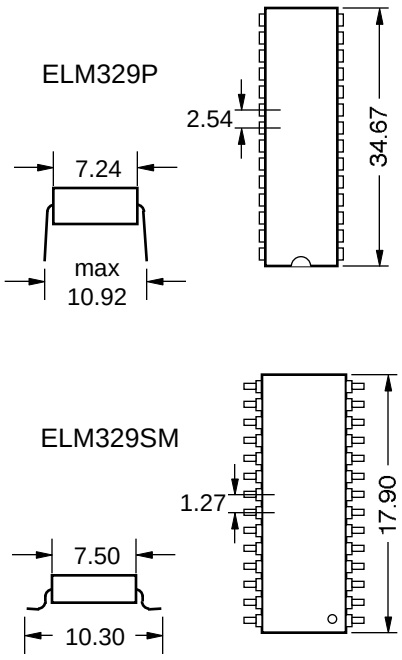
Outline Diagrams

The diagrams at the right show the two package styles that the ELM329 is available in.

The first shows our ELM329P product in what is commonly called a '300 mil skinny DIP package'. It is used for through hole applications.

The ELM329SM package shown at right is also sometimes referred to as 300 mil, and is often called an SOIC package. We have chosen to simply refer to it as an SM (surface mount) package.

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



Note: all dimensions shown are in mm.

Package Drawings and Dimensions Specification
(document name en012702.pdf - 7.5MB)
Go to www.microchip.com, choose 'Design Support' then 'Documentation' and 'Packaging Specifications' or else go to www.microchip.com/packaging then choose either SPDIP or SOIC 28 pin.

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

Ordering Information

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

The ELM329 part numbers are as follows:

300 mil 28 pin Plastic DIP.....ELM329P 300 mil 28 pin SOIC.....ELM329SM

Index

Index (continued)