Host Command Reference

Q and SCL commands for servo and stepper drives

Includes RS-232, RS-485,
Ethernet UDP, Ethernet TCP/IP, EtherNet/IP,
Modbus RTU and Modbus TCP/IP

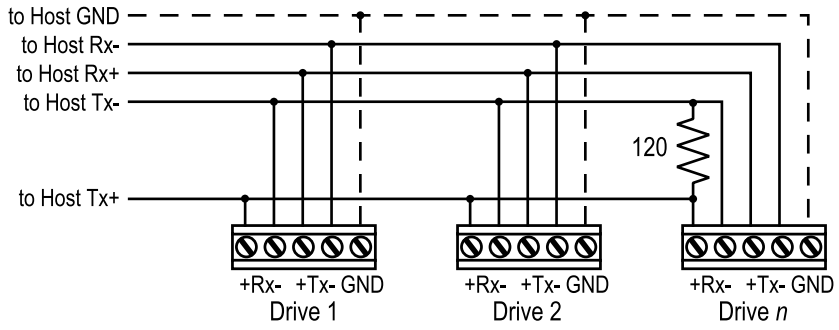
Applied Motion Products
A MOONS' COMPANY

APPLIED MOTION PRODUCTS

920-0002 Rev. W
7/6/2021

Contents

1. Connect the drive TX+ to the host RX+.
2. Connect the drive TX- to the host RX-.
3. Connect the drive RX+ to the host TX+.
4. Connect the drive RX- to the host TX-.
5. Connect GND to the host signal ground.
6. We recommend a 120 ohm terminating resistor be connected between the Rx+ and Rx- terminals of the drive farthest from the host.



NOTE: Proper cable shielding is a must. High voltage, high frequency, high current signals that are present on the servo motor cables can emit a significant amount of electrical interference. Without proper shielding on the communications wiring this interference can disrupt even noise-tolerant differential line drivers.

Getting and Connecting an RS-485 4-wire adapter to your PC.

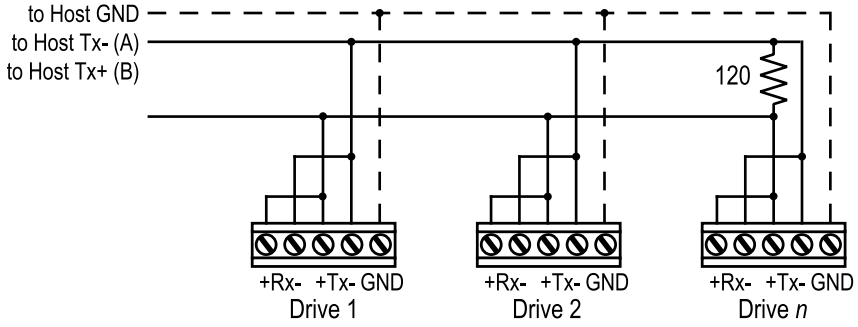
If you are using your computer to communicate to the drive(s) and therefore need an RS-485 adapter, model 117701 from Jameco Electronics (800-831-4242) works well. This adaptor is for a 25-pin serial port. If you are like most people and have a 9-pin serial port on your PC, you will also need to purchase Jameco cable 31721. Connect as follows:

Adaptor Terminal	Drive Terminal
1	RX+
2	RX-
3	TX-
4	TX+

Set the switches on the Jameco adaptor for DCE and TxON, RxON. Don't forget to plug in the DC power adapter that comes with the unit.

Connecting to a host using 2-wire RS-485

An Applied Motion drive's 2-wire RS-485 implementation is a multi-drop network with one pair of wires that is used for both transmit and receive. To make this type of connection you will first need to jumper the TX+ terminal of a drive to it's own RX+ terminal, and then do the same with the TX- and RX- terminals. To then connect a drive to the host, you will need to connect the TX+/RX+ terminals of the drive to the host's TX+/RX+ terminal, and then the TX-/RX- terminals of the drive to the host's TX-/RX- terminal. We also recommend a 120 terminating resistor be connected between the Tx+ and Tx- terminals of the drive farthest from the host. Here is a diagram.



Getting and Connecting an RS-485 2-wire adapter to your PC.

If you are using your computer to communicate to the drive(s) and therefore need an RS-485 adaptor, model 485-25E from Integrity Instruments (800-450-2001) works well. It comes with everything you need. Connect as follows:

Adaptor Terminal	Drive Terminals
A	TX+/RX+
B	TX-/RX-

Before you connect the drive to your system

If you plan to implement a 2-wire or 4-wire RS-485 network of drives, you will first need to address each drive individually. An easy way to do this is prior to hooking the drives up with one of the RS-485 implementations shown above, use the RS-232 cable that came with each drive and the SCL Setup Utility. If you've already connected your drive using one of the RS-485 implementations, completing this sub-section will allow you to test your connections.

First connect your PC and drive. (See preceding sub-sections on connecting to a PC or host for help with this). Then launch the *SCL Setup Utility* on your PC. If you don't have the *SCL Setup Utility* installed, you can get it either from the CD-ROM that came with your drive or from Applied Motion's web site, www.applied-motion.com/support/software.php.

Once the *SCL Setup Utility* is launched, select the proper COM port of your PC, and then apply power to the drive. Press the Caps Lock key on your keyboard (because the drives only accept commands in uppercase). Type RV then press Enter. If the drive has power and is properly wired, it will respond with "RV=x", where x is the firmware version of your drive. This confirms that communication has been established. If you don't see the "RV=x" response, check your wiring and follow the above procedures again.

Next, you must choose an address for each drive. Any of the "low ascii" characters (many of which appear above the number keys on a PC keyboard) are acceptable:

! " # $ % & ' ( ) * + , - . / 0 1 2 3 4 5 6 7 8 9 : ; < > ? @

To find out which address is already in your drive, type DA then press Enter. The drive will respond with "DA=x", where x is the address that was last stored. To change the address, type "DAy", where y is the new address character, then press Enter.

To test the new address, type "yRV" where y is the address you've just assigned to the drive, and then press Enter. For example, if you set the address to % and want to test the address, type "%RV" then press Enter. The drive should respond with "%RV=x" where x is the firmware version of the drive.

Once each drive in your network has been given a unique address, you can proceed with wiring the whole network together.

Appendix D: The PR Command

Because of the intense nature of serial communications required in host mode applications, you are allowed to adjust a drive's serial communications protocol to best fit your application. This adjusting of a drive's serial communications protocol is done using the PR command.

Typically the PR command is used one time when configuring a drive and saved as part of the startup parameters (use SA command to save startup parameters). However, it can be changed at any time to dynamically alter the serial communications.

The PR command works by sending the decimal equivalent of a 9-bit binary "word". Each bit in the word represents a different setting of the serial communications protocol. These settings are additive, meaning when you set a bit to "1", or turn it on, you are adding the functionality of that setting to the serial protocol. Think of this 9-bit word as a bank of 9 dip switches. You can turn each dip switch on or off, and in doing so add or subtract a particular setting from the overall protocol.

The PR command in detail

The diagram to the right shows the assignments of each of the 9 bits in the protocol word. Remember that when you use the PR command the parameter that you send along with the command code (PR) is the decimal equivalent of this binary word. Below are the details of each of the bits and the settings they are assigned to.

Bit 0 - Default ("Standard SCL")

PR cannot be set to 0, so if no other bits in the PR word are set to 1 then at least bit 0 must be set to 1. Setting Bit 0 to 1 when any other bits are also set to 1 has no effect on the communications protocol. For example, PR4 (bit 2 set to 1) is the same as PR5 (bits 0 and 2 set to one). With only bit 0 set to 1, when commands that do not request returned data are received by the drive no other response is sent from the drive. In other words, the drive will only send a response to commands that require a response.

Send data Examples:

Command	Drive Sends	Notes
DI8000	-	Global set distance to 8000 counts or steps
1DI8000	-	Drive with address "1" set distance to 8000 counts or steps

Request data Examples:

Command	Drive Sends	Notes
DI	DI=8000	Global distance request
1DI	1DI=8000	Drive with address "1" responds with distance

Bit 1 - Address Character (always send address character)

With this option set (Bit 1=1) a drive's address character will always be included in the response packet along with any requested data.

Send data Examples:

Command	Drive Sends	Notes
VE50	-	Global set velocity to 50 rps
1VE50	-	Drive with address "1" set velocity to 50 rps

Request data Examples:

Command	Drive Sends	Notes
VE	1VE=50	Drive responds with address “1” and velocity to global velocity request
1VE	1VE=50	Drive responds with address “1” and velocity to specific velocity request from drive at address “1”

Bit 2 - Ack/Nack (always send acknowledge character)

This option causes the drive to acknowledge every transmission from a host, whether the command is requesting data or not. If a host requests data (for example a DI command with no parameter), the response is considered the acknowledgement. However, if the host sends commands that do not request data from the drive, the drive will still respond with one of the following characters:

“%” - The “percent” character is a Normal Acknowledge (Ack) character that means the drive accepted the command and executed it.

“*” - The “asterisk” character is an Exception Acknowledge (Ack) character that means the drive accepted the command and buffered it into the queue. Depending on the status of the queue, execution of the exception acknowledged command(s) can occur at any time after the acknowledge.

“?” - The “question mark” character is a Negative Acknowledge (Nack) character that means a parsing error occurred while the drive was receiving the command. A second character may follow the question mark, which provides an error code describing the type of parsing error. Here is the list of error codes:

Negative Acknowledge Codes

1 Command timed out
2 Parameter is too long
3 Too few parameters
4 Too many parameters
5 Parameter out of range
6 Command buffer (queue) full
7 Cannot process command
8 Program running
9 Bad password
10 Comm port error
11 Bad character
12 I/O point already used by current Command Mode, and cannot be changed (Flex I/O drives only)
13 I/O point configured for incorrect use (i.e., input vs. output) (Flex I/O drives only)
14 I/O point cannot be used for requested function - see HW manual for possible I/O function assignments. (Flex I/O drives only)

Acknowledge characters are always sent out of the RS-232 port. When operating on a 2-wire or 4-wire RS-485 network, the acknowledge characters are sent out under the following conditions:

1. An acknowledge character is sent when the received command has an address character at the beginning.

2. An acknowledge character is NOT sent when global commands (commands without addresses) that do not request data from the drive are used.

3. Global commands that request data will cause data to be returned from the drive(s). This can cause data collisions if there are more than one drive on a network. NOTE: Always use addresses with commands in multi-drop networks to avoid data collisions.

*NOTE: When possible avoid using Acknowledge characters (%, *, ?) as drive addresses in multi-drop networks to prevent confusion.*

Good command Example:

Command	Drive Sends	Notes
DI8000	%	Drive sends normal Ack (over RS-232 port only) in response to global set distance to 8000
1DI8000	1%	Drive at address “1” sends normal Ack (over both ports) in response to address-specific set distance to 8000

Bad command Example:

Command	Drive Sends	Notes
VE200	?5	Drive sends Nack (over RS-232 port only) in response to global set velocity to 200 rps; error code 5 is sent because parameter “200” is out of range
1VE200	1?5	Drive at address “1” sends Nack (over both ports) and error code in response to address-specific set velocity to 200 rps

Buffered command Example:

Command	Drive Sends	Notes
AC10	*	Drive sends Exception Ack (over RS-232 port only) in response to global set acceleration to 10 rps/s
1AC10	1*	Drive at address “1” sends Exception Ack and address (over both ports) in response to address-specific set acceleration

Bit 3 - Checksum

When this bit is 1, checksum is implemented. When this bit is 0, checksum is not implemented.

Bit 4 - RS-485 Adapter mode

Allows using a drive as an RS-232 to RS-485 adapter by letting the host communicate on an RS-485 network through the first drive’s RS-232 port. When the host sends commands with a “~” (tilde) at the beginning of the command to the first drive’s RS-232 port, the command is echoed out of both of that drive’s RS-232 and RS-485 ports. Drives connected on the RS-485 network will receive the same command with the “~” stripped off.

Without the Bit 4 option (Bit 4=0), a drive will normally echo any addressed command out of the RS-232 port only, whether the command was received from the drive’s RS-232 or RS-485 port. What the Bit 4 setting does (Bit 4=1), is force the drive to echo commands out the RS-485 port as well, allowing a host that is connected to a drive through its RS-232 port, to communicate to an RS-485 network of drives.

NOTE: When both Bits 4 and 2 are set (Bit 4=1, Bit 2=1), the host will receive back both the echoed packet and the acknowledge packet. For example, two drives are connected in an RS-485 network, and they both have PR command Bits 4 and 2 set. The first drive, which is also connected to the host via its RS-232 port, is addressed “1”, and the second drive is addressed “2”. Here is what you will see:

Send data Example:

Command	Drive Sends	Notes
~2DI8000	2DI8000	Drive at address “1” echoes original command over both serial ports
	2%	Drive at address “2” responds with ack.

Request data Example:

Command	Drive Sends	Notes
~2DI	2DI	Drive at address “1” echoes original command over both serial ports
	2DI8000	Drive at address “2” responds with distance

Bit 5 - 3-digit numeric register addressing

Each data register in a drive is normally accessed using its single letter, number, or other ascii character. With Bit 5 set (Bit 5=1), each of the data registers is instead accessed with a 3-digit number: 000 to 074. (See the Data Registers section for character and 3-digit numerical assignments). The Bit 5 option implements this specific usage for the RL (Register Load) and RU (Register Upload) commands.

NOTE: When data is returned from a drive (whether Bit 5 is set or not set), the data register is always represented by its single character designation.

RL Command Example:

Command	Drive Sends	Notes
RL017100	-	Load register 017 (“A”) with the value 100
RL017	RLA=100	Drive sends contents of acceleration register

RU command Example:

Command	Drive Sends	Notes
RU0174	RUA=100 RUB=150 RUC=140 RUD=210	Drive responds to register upload command by sending contents of 4 sequential data registers, starting with register 017 (“A”)

Note: 3-digit numeric register addressing may be required for some 3rd party PLCs, HMIs, etc.

Bit 6 - Checksum Type

When this bit is 1, STM type checksum is implemented. When this bit is 0, SSM type checksum is implemented.

SSM type Checksum Protocol

Step-Servo (SSM/TSM/TXM/SS) and SV200 drives support “SSM type” checksum protocol. When bit 3 in PR command is set to 1 and bit 6 in PR is set to 0, for example PR=13, then “SSM type” checksum protocol is implemented.

Step-Servo (SSM/TSM/TXM/SS) and SV200 drives can also support “STM type” checksum protocol. When both bit 3 and bit 6 in PR command are set to 1, for example PR=77, then “STM type” checksum protocol is implemented.

ST/STM/SWM drives do not support “SSM type” checksum protocol.

SSM type Checksum protocol and Data Frame

This Checksum Protocol is based on 1’s complement, and added at the end of every “Command”. There is a delimiter “{” between the “Command” chars and the one byte Checksum. The Data Frame as the following:

Example: “CC” Command

Send Command (from Host):

Host Request: CC{79

Chars	“C”	“C”	“{”	CHKsm: 0x79 (2 ASCII)		“Carriage Return”
Hex	0x43	0x43	0x7B	0x37	0x39	0x0D

Receive back (from Drive):

Drive Response: CC=5{07

Chars	“C”	“C”	“=”	“5”	“{”	CHKsm: 7 (2 ASCII)		“Carriage Return”
Hex	0x43	0x43	0x3D	0x35	0x7B	0x30	0x37	0x0D

If you get a Nack response “?10”, this is a comm port error that may be a bad checksum. To see what it is use the “CE” command which will give back an error code in “Hex”.

Here is the list of Error codes for the “CE” command (the ones in BLUE are for the checksum)

Command	Hex Code	Dec Code	Description
CE	0x0000	0	Communication is normal
	0x0001	1	Communication Parity bit does not match
	0x0002	2	Communication Framing Error
	0x0004	4	noise on the receiver input
	0x0008	8	Overflow error
	0x0010	16	Receive Buffer is Full
	0x0020	32	Transmit Buffer is Full
	0x0040	64	Bad SPI opcode (this only for SPI interface comm)
	0x0080	128	Transmit Time Out
	0x0100	256	Receive Time Out
	0x0200	512	Bad checksum on the receiver
	0x0400	1024	Too many chars for the command

How to using the checksum feature

The checksum is a option feature for all drives. To use this feature, please set the bit3 in PR command.

Example: For Ack (select by Bit2 of PR command) and checksum, set PR13 (Note: the Bit0 should be always set). If you want to know the detail of this command, please refer to the document “Host Command Reference”.

Here is the format of the protocol: [optional] (delimiters)

[Address] Command Parameter(s) (“{“, ASCII 123) **Checksum** (“Carriage Return”, ASCII 13)

- The character “{“, ASCII 123, is used to delimit the main packet and the checksum.
- The checksum value is 8-bit, 1’s complement (add up all the character from the “Address” to “Parameter” don’t include the “{“ character, “mask” off the lower byte (means grabbing the lower byte of the checksum) to restrict it to 8 bits, invert all the bits), and then convert the value to ASCII chars.

NOTE 1: There is no checksum on the “Ack” or “Nack” packets. These are very simple responses that in the case of an “Ack” does not have data or the “Nack” has only an error code.

NOTE 2: when checksumming is turned on, you are **NOT** *allowed to send a packet without the checksum, otherwise it will return back “?12”(means no Checksum), In the other hand, when checksumming is turned off, but send command with checksum, it will return a “?4”(means too many parameters).*

STM type Checksum Protocol

ST/STM/SWM drives support “STM type” checksum protocol. When bit 3 in PR command is set to 1, for example PR=13, then “STM type” checksum protocol is implemented.

Step-Servo (SSM/TSM/TXM/SS) and SV200 drives also support “STM type” checksum protocol. When both bit 3 and bit 6 in PR command are set to 1, for example PR=77, then “STM type” checksum protocol is implemented.

Step-Servo (SSM/TSM/TXM/SS) and SV200 drives can also support “SSM type” checksum protocol. When bit 3 in PR command is set to 1 and bit 6 in PR is set to 0, for example PR=13, then “SSM type” checksum protocol is implemented.

STM type Checksum protocol and Data Frame

This Checksum Protocol is based on 1’s complement, and added at the end of every “Command”. There is a delimiter “{” between the “Command” chars and the one byte Checksum. The Data Frame as the following:

Example: “CC” Command

Send Command (from Host):

Host Request: CC

Chars	“C”	“C”	“{”	CHKsm	“Carriage Return”
Hex	0x43	0x43	0x7B	0x79	0x0D

Receive back (from Drive):

Drive Response: CC=1.2

Chars	“C”	“C”	“=”	“1”	“.”	“2”	“{”	CHKsm	“Carriage Return”
Hex	0x43	0x43	0x3D	0x31	0x2E	0x32	0x7B	0xAB	0x0D

If you get a Nack response “?10”, this is a comm port error that may be a bad checksum. To see what it is use the “CE” command which will give back an error code in “Hex”.

Here is the list of Error codes for the “CE” command (the ones in BLUE are for the checksum)

Command	Hex Code	Dec Code	Description
CE	0x0000	0	Communication is normal
	0x0001	1	Communication Parity bit does not match
	0x0002	2	Communication Framing Error
	0x0004	4	noise on the receiver input
	0x0008	8	Overflow error
	0x0010	16	Receive Buffer is Full
	0x0020	32	Transmit Buffer is Full
	0x0040	64	Bad SPI opcode (this only for SPI interface comm)
	0x0080	128	Transmit Time Out
	0x0100	256	Receive Time Out
	0x0200	512	Bad checksum on the receiver
	0x0400	1024	Too many chars for the command

How to use the checksum feature

The checksum is a optional feature for all drives. To use this feature, please set the bit3 in PR command.

Example: For Ack (select by Bit2 of PR command) and checksum, set PR13 (Note: the Bit0 should be always set).

Here is the format of the protocol: [optional] (delimiters)

[Address] Command Parameter(s) (“{“, ASCII 123) **Checksum** (“Carriage Return”, ASCII 13)

- The character “{“, ASCII 123, is used to delimit the main packet and the checksum.
- The checksum value is 8-bit, 1’s complement (add up all the character from the “Address” to “Parameter” don’t include the “{“ character, “mask” off the lower byte (means grabbing the lower byte of the checksum) to restrict it to 8 bits, invert all the bits), and then convert the value to ASCII chars.

NOTE 1: There is no checksum on the “Ack” or “Nack” packets. These are very simple responses that in the case of an “Ack” does not have data or the “Nack” has only an error code.

NOTE 2: when checksumming is turned on, you are ***NOT*** *allowed to send a packet without the checksum, otherwise it will return back “?12”(means no Checksum), In the other hand, when checksumming is turned off, but send command with checksum, it will return a “?4”(means too many parameters).*

Bit 7 - Little/Big Endian in Modbus Mode

When this bit is 1, Little Endian is used. When this bit is 0, Big Endian is used.

Bit 8 - Full Duplex in RS-422

When this bit is 1, Full Duplex is used. When this bit is 0, Half Duplex is used.

PR Command Examples

Now that you know what the bits in the PR command's 9-bit binary word mean, here are a couple examples showing how you would set the serial communications protocol of your drive.

Example: Turn on Ack/Nack (Bit 2) and 3-digit numeric register addressing function (Bit 5)

The 9-bit word for this combination is - 000100100 - and it's decimal equivalent is 36. Therefore, to set your drive with this serial protocol, you would send the command "PR36" to your drive.

Example: Turn on RS-485 adaptor function (Bit 4)

The 9-bit word for this combination is - 000010000 - and it's decimal equivalent is 16. Therefore, to set your drive with this serial protocol, you would send the command "PR16" to your drive.

Appendix E: Alarm and Status Codes

One of a drive's diagnostic tools is its ability to send alarm and status codes back to a host. The AL (Alarm code) and SC (Status Code) commands can be used by a host to query a drive at any time. If a drive faults or sets an alarm, the AL command allows the host to find out what alarm, or alarms, has been set. Similarly, the SC command allows a host to find out what the status code of a drive is at any time during drive operation. A status code provides information as to whether the drive is running, in position, disabled, homing, and other conditions. Both alarm and status codes can be very useful when initially setting up and integrating a servo system into your machine.

The Alarm and Status codes are hexadecimal equivalents of 16 bit binary "words". Each bit in each binary word is assigned a meaning, and therefore a code word can actually show information about more than one alarm or status condition.

Alarm Code Definitions

AL command

When a host sends the AL command, the response from the drive will be the Hexadecimal equivalent of a 16-bit word. This hexadecimal value is considered the Alarm Code, and the hexadecimal value for each of the bits in the Alarm Code is given below.

Hex Value	BLu	SV	STAC6	ST	STM
0001	*Position Limit*				
0002	CCW Limit				
0004	CW Limit				
0008	*Over Temp*				
0010	*Excess Regen**	*Internal Voltage*	*Excess Regen*	*Internal Voltage*	*Internal Voltage*
0020	*Over Voltage*				
0040	*Under Voltage**	Under Voltage	*Under Voltage*	Under Voltage	Under Voltage
0080	*Over Current*				
0100	*Bad Hall Sensor*		*Open Motor Winding*		
0200	*Bad Encoder*				(not used)
0400	Comm Error				
0800	Bad Flash				
1000	Wizard Failed		No Move		
2000	Current Foldback		Motor Resistance Out of Range	(not used)	(not used)
4000	Blank Q Segment				
8000	No Move		(not used)		

* BLuAC drives only

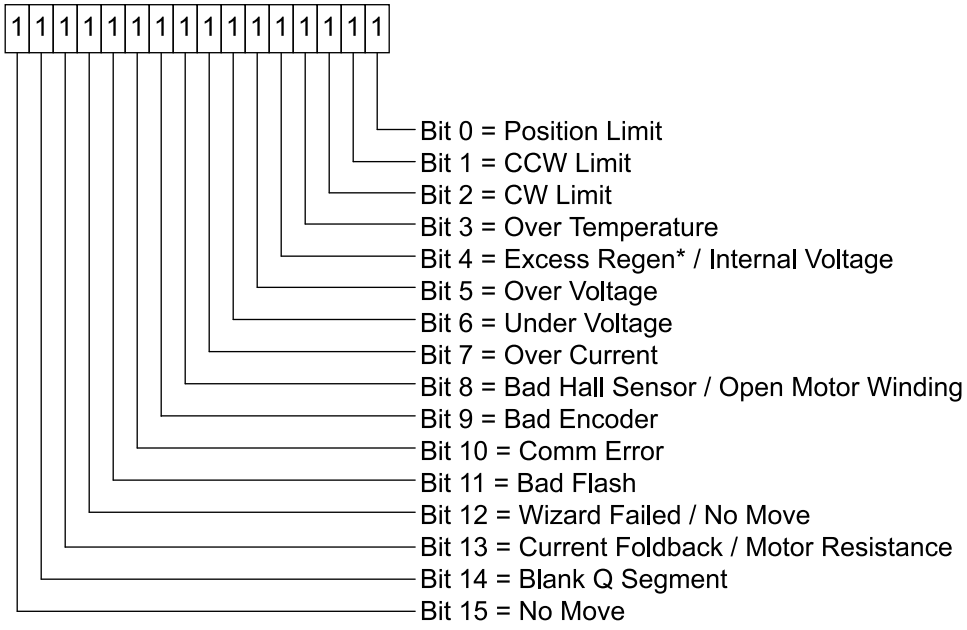
NOTE: Items in *bold italic* represent Drive Faults, which automatically disable the motor. Use the OF command in a Q Program to branch on a Drive Fault.

Example: The drive has hit the CW limit (0004), there is an under voltage condition (0040), and an encoder wiring connection has been lost resulting in an encoder fault (0200). The resulting Alarm Code is 0244, and when the host sends the "AL" command the drive will respond with "AL=244".

"f" data register

Another way to retrieve the Alarm Code is to use the "f" data register. If the host sends the RLf command, the response from the drive will be the decimal equivalent of the 16-bit Alarm Code word. The diagram below

shows the 16 bit assignments for the Alarm Code (which of course match the hexadecimal values above).



Alarm Code 16-bit word

Example: The drive has hit the CW limit (bit 2), there is an under voltage condition (bit 6), and an encoder wiring connection has been lost resulting in an encoder fault (bit 9). The resulting Alarm Code binary word is 0000 0010 0100 0100. The decimal equivalent of this word is 580, so the response from the drive to the RLf command will be “RLf=580”.

Status Code Definitions

SC command

When a host sends the SC command, the response from the drive will be the Hexadecimal equivalent of a 16-bit word. This hexadecimal value is considered the Status Code, and the hexadecimal value for each of the bits in the Status Code is given below. When a host sends the SC command, the response from the drive will actually be the Hexadecimal equivalent of this 16-bit word. This hexadecimal value is considered the Status Code, and the equivalent hexadecimal value for each of the bits is given below.

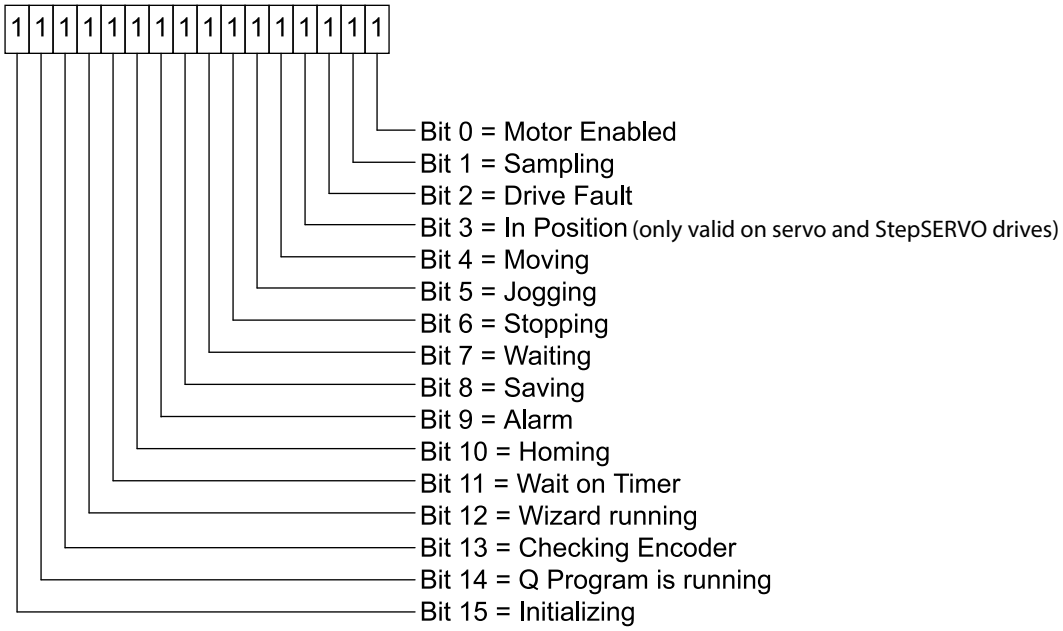
Hex Value	Status Code bit definition
0001	Motor Enabled (Motor Disabled if this bit = 0)
0002	Sampling (for Quick Tuner)
0004	Drive Fault (check Alarm Code)
0008	In Position (motor is in position), only valid on servo and StepSERVO drives
0010	Moving (motor is moving)
0020	Jogging (currently in jog mode)
0040	Stopping (in the process of stopping from a stop command)
0080	Waiting for an input (executing WI command)
0100	Saving (parameter data is being saved)
0200	Alarm present (check Alarm Code)
0400	Homing (executing an SH command)
0800	Wait Time (executing a WT command)

Hex Value	Status Code bit definition
1000	Wizard running (Timing Wizard is running)
2000	Checking encoder (Timing Wizard is running)
4000	Q Program is running
8000	Initializing (happens at power up) ; Servo Ready (for SV200 only)

Example: The drive is running a stored Q program (hex value 4000), it's in position (hex value 0008), and it's waiting for the input specified by the WI command (hex value 0080). The Status Code for this condition is 4088, and when the host sends the "SC" command the drive will respond with "SC=4088".

"s" data register

Another way to retrieve the Status Code is to use the "s" data register. If the host sends the RLs command, the response from the drive will be the decimal equivalent of the 16-bit Status Code word. The diagram below shows the 16 bit assignments for the Status Code (which of course match the hexadecimal values above).



Status Code 16-bit word

Example: The drive is running a stored Q program (bit 14), it's in position (bit 3), and it's waiting for the input specified by the WI command (bit 7). The resulting Status Code binary word is 0100 0000 1000 1000. The decimal equivalent of this word 16,520, so the response from the drive to the RLs command will be "RLs=16520".

A useful tool for converting between binary, decimal, and hexadecimal.

If you're using a Windows-based PC as a host with your drive (which you'll definitely be doing at some point during the project), you can use the Calculator utility that comes with Windows to convert Alarm and Status Codes between binary, decimal, and hexadecimal values. This utility is usually found in Start Menu, Programs, Accessories. Once open, make sure to choose Scientific view from the View menu of Calculator. This view provides radio buttons for Hex, Dec, and Bin.

To figure out what your Alarm or Status Code is telling you, first select the appropriate radio button (Hex for the AL or SC commands, Dec for the "f" and "s" registers), then enter the response from the drive. Now you can toggle between Hex, Dec, and Bin to compare the values to the tables and diagrams above. Note: Calculator does not show leading zeros in a binary number, so you may see less than 16 bits when you select Bin. That's OK, just start counting from the right with Bit 0 and you'll be able to determine the conditions set in the codes.

LED and 7-Segment Display codes

In addition to the Alarm and Status codes, most drive alarms and faults as well as some status codes are displayed at the front of the drives, via either two-color flashing LED codes or 7-segment LED codes. The following tables show the various codes available with Applied Motion drives.

BLuDC LED codes

Items in ***bold italic*** are Drive Faults.

		DESCRIPTION
-	solid	Motor disabled
-	flashing slowly	Motor enabled
-	flashing quickly	Q program running (Q drives only)
1	1	***position limit***
1	2	move attempted while disabled
2	1	CCW limit
2	2	CW limit
3	1	***over temperature (> 85 deg C)***
3	2	bad flash
4	1	***over voltage (> 55 Vdc)***
4	2	under voltage (< 18 Vdc)
5	1	***over current / short circuit***
5	2	current limit
6	1	***bad hall***
6	2	***bad encoder***
7	1	serial communication error

For SV200 series alarm codes, see SV200 Hardware Manual

BLuAC 7-Segment codes

Items in *bold italic* are Drive Faults.

	Position Mode		*Over Temp*		Comm Error
	Velocity Mode		*Over Voltage*		Move attempted while disabled
	Torque Mode		*Under Voltage*		Drive Start-up
	Step Mode		*Over Current*		Bad Flash
	Si Mode		Current Limit		Comm Time-out
	Drive Disabled		*Hall Bad*		Stack Overflow
	*Position Limit*		*Bad Encoder*		Stack Underflow
	CCW Limit		Memory Failed		Q Program Running
	CW Limit		*Excess Regen*		Drive enabled when flashing

SV LED codes

Items in ***bold italic*** are Drive Faults.

		DESCRIPTION
-	solid	Motor disabled
-	flashing slowly	Motor enabled
-	flashing quickly	Q program running (Q drives only)
1	1	***position limit***
1	2	move attempted while disabled
2	1	CCW limit
2	2	CW limit
3	1	***over temp***
3	2	***internal voltage out of range***
3	3	attempt to load blank Q segment
4	1	***over voltage***
4	2	under voltage
4	3	***Bad Si program instruction***
5	1	***over current / short circuit***
5	2	current limit
6	1	***bad hall***
6	2	***bad encoder***
7	1	serial communication error
7	2	bad flash

STAC6 LED codes

Items in ***bold italic*** are Drive Faults.

		DESCRIPTION
-	solid	Motor disabled
-	flashing slowly	Motor enabled
-	flashing quickly	Q program running (Q drives only)
1	1	***motor stall (w/ optional encoder only)***
1	2	move attempted while disabled
2	1	CCW limit
2	2	CW limit
3	1	***over temp***
3	2	***excess regen***
4	1	***over voltage***
4	2	***under voltage***
4	3	***Bad Si program instruction***
5	1	***over current / short circuit***
5	2	motor resistance out of range

6	1	***open motor winding***
6	2	***bad encoder signal (w/ optional encoder only)***
7	1	serial communication error

ST-Q/Si LED codes

Items in ***bold italic*** are Drive Faults.

		DESCRIPTION
-	solid	Motor disabled
-	flashing slowly	Motor enabled
-	flashing quickly	Q program running (Q drives only)
1	1	***motor stall (w/ optional encoder only)***
1	2	move attempted while disabled
2	1	CCW limit
2	2	CW limit
3	1	***over temp***
3	2	***internal voltage out of range***
4	1	***over voltage***
4	2	under voltage
4	3	***Bad Si program instruction***
5	1	***over current / short circuit***
6	1	***open motor winding***
6	2	***bad encoder signal (w/ optional encoder only)***
7	1	serial communication error

ST-S LED codes

Items in ***bold italic*** are Drive Faults.

		DESCRIPTION
-	solid	Motor disabled
-	flashing slowly	Motor enabled
1	2	move attempted while disabled
2	1	CCW limit
2	2	CW limit
3	1	***over temp***
3	2	***internal voltage out of range***
4	1	***over voltage***
4	2	under voltage
5	1	***over current / short circuit***
6	1	***open motor winding***
7	1	serial communication error

STM / SSM / STM / SWM / TSM / TXM LED codes

Items in ***bold italic*** are Drive Faults.

		DESCRIPTION
-	solid	Motor disabled
-	flashing slowly	Motor enabled
-	flashing quickly	Q program running (Q drives only)
1	1	***motor stall (w/ optional encoder only)***
1	2	move attempted while disabled
2	1	CCW limit
2	2	CW limit
3	1	***over temp***
3	2	***internal voltage out of range***
4	1	***over voltage***
4	2	under voltage
5	1	***over current / short circuit***
6	1	***open motor winding***
7	1	serial communication error

Appendix F: Working with Inputs and Outputs

This Appendix covers I/O usage on drives from Applied Motion Products.

Low v. High

When working with inputs and outputs it is important to remember the designations **low** and **high**. If current is flowing into or out of an input or output the logic state for that input/output is defined as **low** or closed. If no current is flowing, or the input/output is not connected, the logic state is **high** or open. A low state is represented by the "L" character in parameters of commands that affect inputs/outputs. For example, WIX4L means "wait for input X4 low", and SO1L means "set output 1 low". A high state is represented by the "H" character.

When working with the analog inputs, "L" designates an analog value lower than the value set by the AT command. Similarly "H" designates an analog value greater than the value set by the AT command.

"X" Marks The Spot

When using a dual input command, both I/O points used must reside on the same connector. That is, if an "X" input such as X2 is used for the first input, the second input is assumed to use an "X" as well since it must reside on the same connector. Since it is not possible to mix I/O from different banks, there is no need for the "X" character on the second I/O point. See the "Parameter Details" section in the tables below for specific details.

Parameter Details

The following tables show general I/O details for commands as they relate to specific drives. There are exceptions to these general rules, so be sure to check the command pages for the specific SCL commands you wish to implement, as well as the list of exceptions at the end of this section. For specific voltage or wiring questions, consult your drive's hardware manual.

Input Parameter Details

BLu-S, BLu-Q
STAC6-S, STAC6-Q, STAC6-C

Parameter #1	Optional "X", input number, input condition NOTE: Including/omitting the optional "X" has no effect on the execution of the command.
- units	Optional "X", integer, letter
- range	- integer: 0 (encoder index, if present), 1 - 7, 8 (Analog Command), 9 (AIN1), : (AIN2) - letter: L = Low, H = High, F = Falling Edge, R = Rising Edge
Parameter #2	Input number, input condition
- units	integer, letter
- range	- integer: 0 (encoder index, if present), 1 - 7, 8 (Analog Command), 9 (AIN1), : (AIN2) - letter: L = Low, H = High, F = Falling Edge, R = Rising Edge

BLu-SE, BLu-QE, BLu-Si
STAC6-SE, STAC6-QE, STAC6-Si

Parameter #1	Optional “X”, input number, input condition NOTE: Including the optional “X” indicates that the input(s) resides on the IN/OUT1 or main drive board connector. Omitting the “X” indicates that the input(s) resides on the IN/OUT2 or top board connector.
- units	Optional “X”, integer, letter
- range	- integer for IN/OUT1 or main drive board connector: X0 (encoder index, if present), X1 - X7, X8 (Analog Command), X9 (AIN1), X: (AIN2) -integer for IN/OUT2 or top board connector: 1 - 8 - letter: L = Low, H = High, F = Falling Edge, R = Rising Edge
Parameter #2	Input number, input condition
- units	integer, letter
- range	- integer for IN/OUT1 or main drive board connector: 0 (encoder index, if present), 1 - 7, 8 (Analog Command), 9 (AIN1), : (AIN2) -integer for IN/OUT2 or top board connector: 1 - 8 - letter: L = Low, H = High, F = Falling Edge, R = Rising Edge

STAC5-S, SVAC3-S

Parameter #1	Optional “X”, input number, input condition NOTE: Including/omitting the optional “X” has no effect on the execution of the command.
- units	integer, letter
- range	- integer: 0 (encoder index, if present), 1 - 4, 8 (AIN) - letter: L = Low, H = High, F = Falling Edge, R = Rising Edge
Parameter #2	input number, input condition
- units	integer, letter
- range	- integer: 0 (encoder index, if present), 1 - 4, 8 (AIN) - letter: L = Low, H = High, F = Falling Edge, R = Rising Edge

STAC5-Q, STAC5-IP
SVAC3-Q, SVAC3-IP

Parameter #1	Optional “X”, input number, input condition NOTE: Including the optional “X” indicates that the input(s) resides on the IN/OUT1 connector (DB-15). Omitting the “X” indicates that the input(s) resides on the OPT2 connector (DB-25).
- units	Optional “X”, integer, letter
- range	- integer for IN/OUT1 connector: X0 (encoder index, if present), X1 - X4, X8 (AIN) - integer for OPT2 connector: 1 - 8 - letter: L = Low, H = High, F = Falling Edge, R = Rising Edge
For those commands with Parameter #2	input number, input condition
- units	integer, letter
- range	- integer for IN/OUT1 connector: 0 (encoder index, if present), 1 - 4, 8 (AIN) - integer for OPT2 connector: 1 - 8 - letter: L = Low, H = High, F = Falling Edge, R = Rising Edge

ST-Q, ST-Si, ST-C, ST-IP
SV7-S, SV7-Q, SV7-Si, SV7-C, SV7-IP

Parameter #1	Optional “X”, input number, input condition NOTE: Including/omitting the optional “X” has no effect on the execution of the command.
- units	integer, letter
- range	- integer: 0 (encoder index, if present), 1 - 8, 9 (Analog Command), : (AIN1), ; (AIN2) - letter: L = Low, H = High, F - Falling Edge, R = Rising Edge
Parameter #2	input number, input condition
- units	integer, letter
- range	- integer: 0 (encoder index, if present), 1 - 8, 9 (Analog Command), : (AIN1), ; (AIN2) - letter: L = Low, H = High, F = Falling Edge, R = Rising Edge

ST-S, ST-Plus
STM17S, STM17Q
STM23S, STM23Q
SSM23IP
STM/SWM/TXM24S
STM/SWM/TXM24Q

Parameter #1	Optional “X”, input number, input condition NOTE: Including/omitting the optional “X” has no effect on the execution of the command.
- units	optional “X”, integer, letter
- range	- integer: 0 (encoder index, if present), 1 (STEP), 2 (DIR), 3 (EN), 4 (AIN) - letter: L = Low, H = High, F = Falling Edge, R = Rising Edge
Parameter #2	input number, input condition
- units	integer, letter
- range	- integer: 0 (encoder index, if present), 1 (STEP), 2 (DIR), 3 (EN), 4 (AIN) - letter: L = Low, H = High, F = Falling Edge, R = Rising Edge

STM17C

Parameter #1	Optional “X”, input number, input condition NOTE: Including/omitting the optional “X” has no effect on the execution of the command.
- units	Optional “X”, integer, letter
- range	- integer: 0 (encoder index, if present), 1 - 3, 4 (AIN) - letter: L = Low, H = High, F = Falling Edge, R = Rising Edge
Parameter #2	input number, input condition
- units	integer, letter
- range	- integer: 0 (encoder index, if present), 1 - 3, 4 (AIN) - letter: L = Low, H = High, F = Falling Edge, R = Rising Edge

STM23C
STM24C

Parameter #1	Optional “X”, input number, input condition NOTE: Including/omitting the optional “X” has no effect on the execution of the command.
- units	Optional “X”, integer, letter
- range	- integer: 0 (encoder index, if present), 1 - 3 - letter: L = Low, H = High, F - Falling Edge, R = Rising Edge
Parameter #2	input number, input condition
- units	integer, letter
- range	- integer: 0 (encoder index, if present), 1 - 3 - letter: L = Low, H = High, F = Falling Edge, R = Rising Edge

STM24SF, STM24QF

Drives with Flex I/O allow a user to configure I/O1 through I/O4 as either inputs or outputs by using the Set Direction (SD) command.

Parameter #1	Optional “X”, input number, input condition NOTE: Including/omitting the optional “X” has no effect on the execution of the command.
- units	Optional “X”, integer, letter
- range	- integer: 0 (encoder index, if present), 1 - 4, 5 (AIN) - letter: L = Low, H = High, F - Falling Edge, R = Rising Edge
Parameter #2	input number, input condition
- units	integer, letter
- range	- integer: 0 (encoder index, if present), 1 - 4, 5 (AIN) - letter: L = Low, H = High, F = Falling Edge, R = Rising Edge

Exceptions:

- When using the Follow Encoder or Hand Wheel commands (FE or HW, respectively), the master encoder channels A and B must be wired to drive inputs STEP/X1/IN1 and DIR/X2/IN2. In these modes, these inputs must not be used for sensor inputs.
- Using the On Input (OI) command with no parameter will disable the interrupt function.
- The Seek Home (SH) command makes use of the drive's CW and CCW limit functions. As such, the home sensor may not be wired to the following inputs:

STAC5-S:	X1, X2	STM17-S/Q:	STEP, DIR
STAC5-Q/IP:	IN7, IN8	STM17-C:	IN1, IN2
SVAC3-S:	X1, X2	STM23-S/Q:	STEP, DIR
SVAC3-Q/IP:	IN7, IN8	STM23-C:	IN1, IN2
BLu:	X6, X7	STM24-SF/QF:	I/O3, I/O4
STAC6:	X6, X7	STM24-C:	IN1, IN2
ST-S/Plus:	STEP, DIR		
ST-Q/Si/C/IP:	X7, X8		
SV7:	X7, X8		

Output Parameter Details

For SV200 series, see SV200 Hardware Manual and SVX Servo Suite software

BLu-S, BLu-Q
STAC6-S, STAC6-Q, STAC6-C

Parameter #1	Optional "Y", output number, output condition NOTE: Including/omitting the optional "Y" has no effect on the execution of the command.
- units	Optional "Y", integer, letter
- range	- integer: 1 - 3 - letter: L = Low, H = High

BLu-SE, BLu-QE, BLu-Si
STAC6-SE, STAC6-QE, STAC6-Si

Parameter #1	Optional "Y", output number, output condition NOTE: Including the optional "Y" indicates that the output(s) resides on the IN/OUT1 or main drive board connector. Omitting the "Y" indicates that the output(s) resides on the IN/OUT2 or top board connector.
- units	Optional "Y", integer, letter
- range	- integer for IN/OUT1 or main drive board connector: Y1 - Y3 - integer for IN/OUT2 or top board connector: 1 - 4 - letter: L = Low, H = High

STAC5-S
SVAC3-S

Parameter #1	Optional “Y”, output number, output condition NOTE: Including/omitting the optional “Y” has no effect on the execution of the command.
- units	Optional “Y”, integer, letter
- range	- integer: 1 - 2 - letter: L = Low, H = High

STAC5-Q, STAC5-IP
SVAC3-Q, SVAC3-IP

Parameter #1	Optional “Y”, output number, output condition NOTE: Including the optional “Y” indicates that the output(s) resides on the IN/OUT1 connector (DB-15). Omitting the “Y” indicates that the output(s) resides on the OPT2 connector (DB-25).
- units	Optional “Y”, integer, letter
- range	- integer for IN/OUT1 connector: Y1- Y2 - integer for OPT2 connector: 1 - 4 - letter: L = Low, H = High

ST-Q, ST-Si, ST-C, ST-IP
SV7-S-SV7-Q, SV7-Si, SV7-C, SV7-IP

Parameter #1	Optional “Y”, output number, output condition NOTE: Including/omitting the optional “Y” has no effect on the execution of the command.
- units	Optional “Y”, integer, letter
- range	- integer: 1 - 4 - letter: L = Low, H = High

ST-S, ST-Plus
STM17S, STM17Q, STM17C
STM23S, STM23Q, STM23C
STM24C, SSM23IP, SWM,
TSM, TXM

Parameter #1	Optional “Y”, output number, output condition NOTE: Including/omitting the optional “Y” has no effect on the execution of the command.
- units	Optional “Y”, integer, letter
- range	- integer: 1 - letter: L = Low, H = High

STM24SF, STM24QF

Drives with Flex I/O allow a user to configure I/O1 through I/O4 as either inputs or outputs by using the Set Direction (SD) command.

Parameter #1	Optional “Y”, output number, output condition NOTE: Including/omitting the optional “Y” has no effect on the execution of the command.
- units	Optional “Y”, integer, letter
- range	- integer: 1 - 4 - letter: L = Low, H = High, F - Falling Edge, R = Rising Edge

Appendix G: eSCL (SCL over Ethernet) Reference

Introduction

eSCL is Applied Motion Products' language for commanding and querying motion control products over Ethernet. It is supported by several motion control devices, including the ST5-Q-E, ST10-Q-E and SV7-Q-E. In addition to sending commands to a drive from a host in real time, you can also use our Q Programmer software to embed sequences of commands, called Q Programs, in a drive. These programs can be set to execute automatically at power up, or can be triggered by commands sent from the host.

This guide is intended to help you connect and configure your drive and to help you start writing your own eSCL host application.

Getting Started

There are three steps required to create an eSCL application with your new Applied Motion Products motor driver. Each of these is explained in a separate section of this manual.

- Connect the drive to your PC. This includes getting the drive physically connected to your network (or directly to the PC), setting the drive's IP address, and setting the appropriate networking properties on your PC.
- Configure the drive for your motor and application. For step motor drives, you'll need to use a suitable version of our Configurator software. For servos, use Quick Tuner.
- Create your own application. This guide includes code examples in Visual Basic and C# to help you get started. You can download the example in their entirety, from our website, but we recommend reading the explanations in the guide first.

Connecting a Drive to Your PC

This process requires three steps

- Get the drive physically connected to your network (or directly to the PC)
- Set the drive's IP address
- Set the appropriate networking properties on your PC.

Addresses, Subnets, and Ports

Every device on an Ethernet network must have a unique IP address. In order for two devices to communicate with each other, they must both be connected to the network and they must have IP addresses that are on the same subnet. A subnet is a logical division of a larger network. Members of one subnet are generally not able to communicate with the members of another. Subnets are defined by the choices of IP addresses and subnet masks.

If you want to know the IP address and subnet mask of your PC, select Start…All Programs…Accessories…Command Prompt. Then type "ipconfig" and press Enter. You should see something like this:

Point of Interest

AMP recommends performing all Ethernet configuration of the drive while connected directly to a PC via a CAT-5 Ethernet cable. This avoids many potential communication problems associated with frequent IP address changes on a larger network.

Once fully configured, the drive may be used on a plant network without issue.

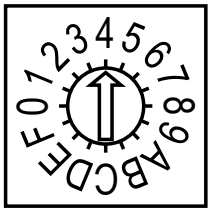
See the section titled "ARP Tables - the Ghost in the Machine" below for further information.

```
Windows IP Configuration

Ethernet adapter Local Area Connection:

        Connection-specific DNS Suffix  . :
        IP Address. . . . . . . . . . . . : 192.168.0.22
        Subnet Mask . . . . . . . . . . . : 255.255.255.0
        Default Gateway . . . . . . . . . : 192.168.0.254
```

If your PC's subnet mask is set to 255.255.255.0, a common setting known as a Class C subnet mask, then your machine can only talk to another network device whose IP address matches yours in the first three octets. (The numbers between the dots in an IP address are called octets.) For example, if your PC is on a Class C subnet and has an IP address of 192.168.0.20, it can talk to a device at 192.168.0.40, but not one at 192.168.1.40. If you change your subnet mask to 255.255.0.0 (Class B) you can talk to any device whose first two octets match yours. Be sure to ask your system administrator before doing this. You network may be segmented for a reason.

Your drive includes a 16 position rotary switch for setting its IP address. The factory default address for each switch setting is shown in the table to the right.

IP Address*	
0	10.10.10.10
1	192.168.1.10
2	192.168.1.20
3	192.168.1.30
4	192.168.0.40
5	192.168.0.50
6	192.168.0.60
7	192.168.0.70
8	192.168.0.80
9	192.168.0.90
A	192.168.0.100
B	192.168.0.110
C	192.168.0.120
D	192.168.0.130
E	192.168.0.140
F	DHCP



Settings 1 through E can be changed using the STAC Configurator software (use Quick Tuner for servo drives). Setting 0 is always "10.10.10.10", the universal recovery address. If someone were to change the other settings and not write it down or tell anyone (I'm not naming names here, but you know who I'm talking about) then you will not be able to communicate with your drive. The only way to "recover" it is to use the universal recovery address.

Setting F is "DHCP", which commands the drive to get an IP address from a DHCP server on the network. The IP address automatically assigned by the DHCP server may be "dynamic" or "static" depending on how the administrator has configured DHCP. The DHCP setting is reserved for advanced users.

Your PC, or any other device that you use to communicate with the drive, will also have a unique address.

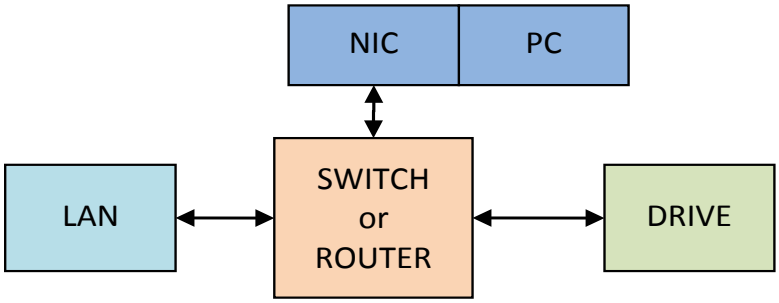
On the drive, switch settings 1 through E use the standard class B subnet mask (i.e. "255.255.0.0"). The mask for the universal recovery address is the standard class A (i.e. "255.0.0.0").

One of the great features of Ethernet is the ability for many applications to share the network at the same time. Ports are used to direct traffic to the right application once it gets to the right IP address. The UDP eSCL port in our drives is 7775. To send and receive commands using TCP, use port number 7776. You'll need to know this when you begin to write your own application. You will also need to choose an open (unused) port number for your application. Our drive doesn't care what that is; when the first command is sent to the drive, the drive will make note of the IP address and port number from which it originated and direct any responses there. The drive will also refuse any traffic from other IP addresses that is headed for the eSCL port. The first application to talk to a drive "owns" the drive. This lock is only reset when the drive powers down.

If you need help choosing a port number for your application, you can find a list of commonly used port numbers at http://www.iana.org/assignments/port-numbers.

One final note: Ethernet communication can use one or both of two "transport protocols": UDP and TCP. eSCL commands can be sent and received using either protocol. UDP is simpler and more efficient than TCP, but TCP is more reliable on large or very busy networks where UDP packets might occasionally be dropped.

Option 1: Connect a Drive to Your Local Area Network



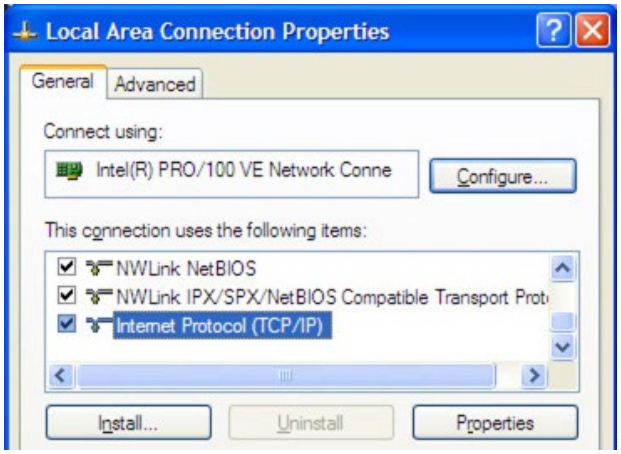
If you have a spare port on a switch or router and if you are able to set your drive to an IP address that is compatible with your network, and not used by anything else, this is a simple way to get connected. This technique also allows you to connect multiple drives to your PC. If you are on a corporate network, please check with your system administrator before connecting anything new to the network. He or she should be able assign you a suitable address and help you get going.

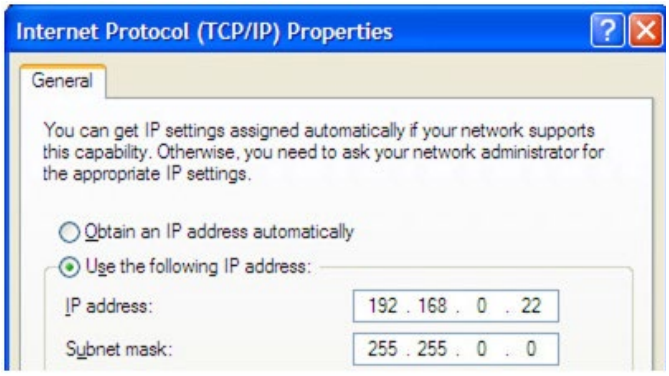
If you are not sure which addresses are already used on your network, you can find out using “Angry IP scanner”, which can be downloaded free from http://www.angryip.org/w/Download. But be careful: an address might appear to be unused because a computer or other device is currently turned off. And many networks use dynamic addressing where a DHCP server assigns addresses “on demand”. The address you choose for your drive might get assigned to something else by the DHCP server at another time.

Once you’ve chosen an appropriate IP address for your drive, set the rotary switch according the address table above. If none of the default addresses are acceptable for your network, you can enter a new table of IP addresses using *Configurator*. If your network uses addresses starting with 192.168.0, the most common subnet, you will want to choose an address from switch settings 4 through E. Another common subnet is 192.168.1. If your network uses addresses in this range, the compatible default selections are 1, 2 and 3.

If your PC address is not in one of the above private subnets, you will have to change your subnet mask to 255.255.0.0 in order to talk to your drive. To change your subnet mask:

1. On Windows XP, right click on “My Network Places” and select properties. On Windows 7, click Computer. Scroll down the left pane until you see “Network”. Right click and select properties. Select “Change adapter settings”
2. You should see an icon for your network interface card (NIC). Right click and select properties.
3. Scroll down until you see “Internet Properties (TCP/IP)”. Select this item and click the Properties button. On Windows 7 and Vista, look for “(TCP/IPv4)”



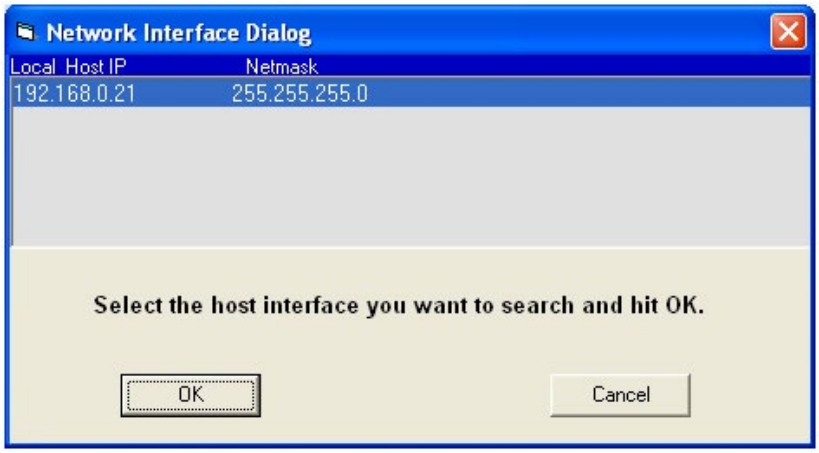


4. If the option “Obtain an IP address automatically” is selected, your PC is getting an IP address and a subnet mask from the DHCP server. Please cancel this dialog and proceed to the next section of this manual: “Using DHCP”.
5. If the option “Use the following IP address” is selected, life is good. Change the subnet mask to “255.255.0.0” and click OK.

Using DCHP

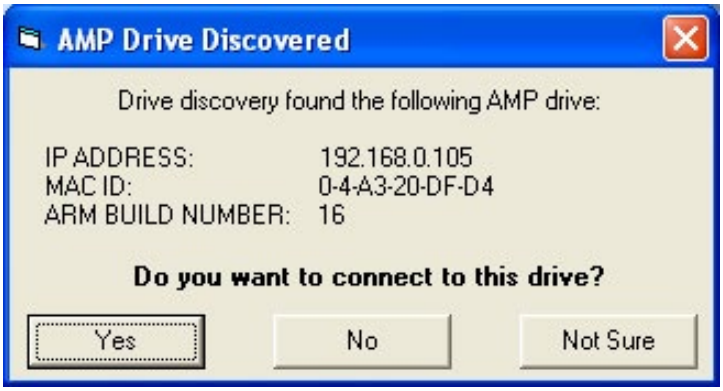
If you want to use your drive on a network that where all or most of the devices use dynamic IP addresses supplied by a DHCP server, set the rotary switch to “F”. When the drive is connected to the network and powered on, it will obtain an IP address and a subnet mask from the server that is compatible with your PC. The only catch is that you won’t know what address the server assigns to your drive. Ethernet Configurator can find your drive using the Drive Discovery feature, as long as your network isn’t too large. With the drive connected to the network and powered on, select Drive Discovery from the Drive menu.

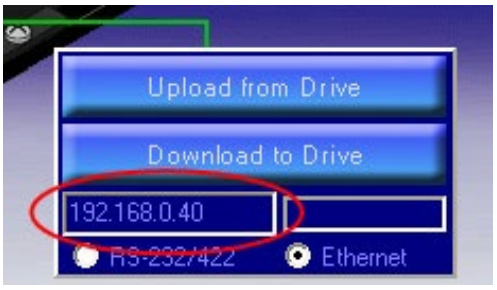
You will see a dialog such as this:



Normally, Drive Discovery will only detect one network interface card (NIC), and will select it automatically. If you are using a laptop and have both wireless and wired network connections, a second NIC may appear. Please select the NIC that you use to connect to the network to which you’ve connected your drive. Then click OK. Drive Discovery will notify you as soon as it has detected a drive.

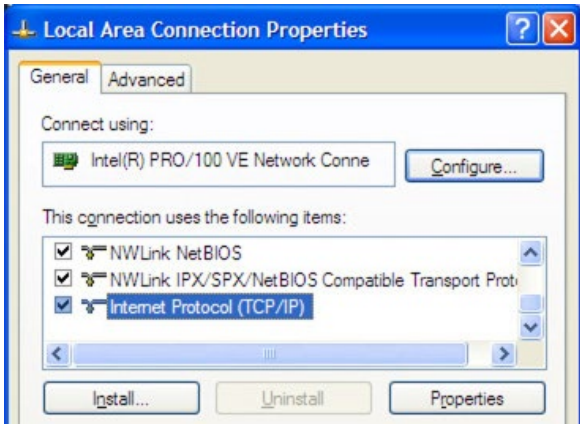
If you think this is the correct drive, click Yes. If you’re not sure, click Not Sure and Drive Discovery will look for additional drives on you network. Once you’ve told Drive Discovery which drive is yours, it will automatically enter that drive’s IP address in the IP address text box so that you are ready to communicate.



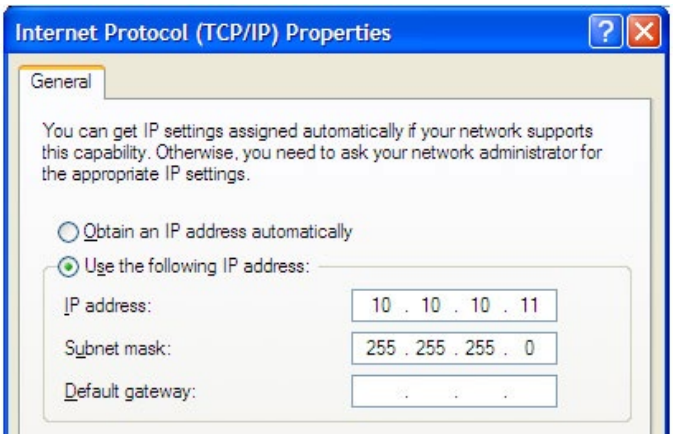


Option 2: Connect a Drive Directly to Your PC

1. Connect one end of a CAT5 Ethernet cable into the LAN card (NIC) on your PC and the other into the drive.
2. Set the IP address on the drive to “10.10.10.10” by setting the rotary switch at “0”.
3. To set the IP address of your PC:
   a. On Windows XP, right click on “My Network Places” and select properties.
   b. On Windows 7, click Computer. Scroll down the left pane until you see “Network”. Right click and select properties. Select “Change adapter settings”
4. You should see an icon for your network interface card (NIC). Right click and select properties.
   a. Scroll down until you see “Internet Properties (TCP/IP)”. Select this item and click the Properties button.
   b. On Windows 7 and Vista, look for “(TCP/IPv4)”



5. Select the option “Use the following IP address”. Then enter the address “10.10.10.11”. This will give your PC an IP address that is on the same subnet as the drive. Windows will know to direct any traffic intended for the drive’s IP address to this interface card.
6. Next, enter the subnet mask as “255.255.255.0”.
7. Be sure to leave “Default gateway” blank. This will prevent your PC from looking for a router on this subnet.

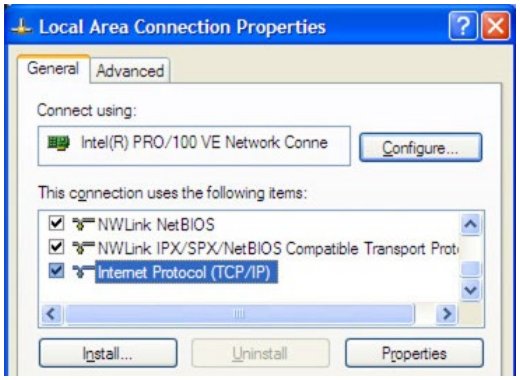


8. Because you are connected directly to the drive, anytime the drive is not powered on your PC may annoy you with a small message bubble in the corner of your screen saying “The network cable is unplugged.”

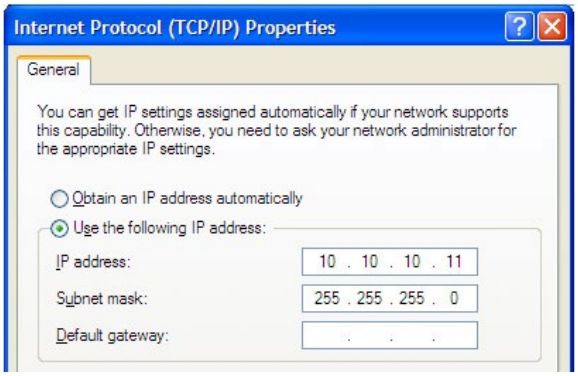
Option 3: Use Two Network Interface Cards (NICs)

This technique allows you to keep your PC connected to your LAN, but keeps the drive off the LAN, preventing possible IP conflicts or excessive traffic.

1. If you use a desktop PC and have a spare card slot, install a second NIC and connect it directly to the drive using a CAT5 cable. You don't need a special "crossover cable"; the drive will automatically detect the direct connection and make the necessary physical layer changes.
2. If you use a laptop and only connect to your LAN using wireless networking, you can use the built-in RJ45 Ethernet connection as your second NIC.
3. Set the IP address on the drive to "10.10.10.10" by setting the rotary switch at "0".
4. To set the IP address of the second NIC:
   a. On Windows XP, right click on "My Network Places" and select properties.
   b. On Windows 7, click Computer. Scroll down the left pane until you see "Network". Right click and select properties. Select "Change adapter settings"



5. You should see an icon for your newly instated NIC. Right click again and select properties.
   a. Scroll down until you see "Internet Properties (TCP/IP)". Select this item and click the Properties button.
   b. On Windows 7 and Vista, look for "(TCP/IPv4)"
6. Select the option "Use the following IP address". Then enter the address "10.10.10.11". This will give your PC an IP address that is on the same subnet as the drive. Windows will know to direct any traffic intended for the drive's IP address to this interface card.
7. Next, enter the subnet mask as "255.255.255.0". Be sure to leave "Default gateway" blank. This will prevent your PC from looking for a router on this subnet.



8. Because you are connected directly to the drive, anytime the drive is not powered on your PC will annoy you with a small message bubble in the corner of your screen saying "The network cable is unplugged."

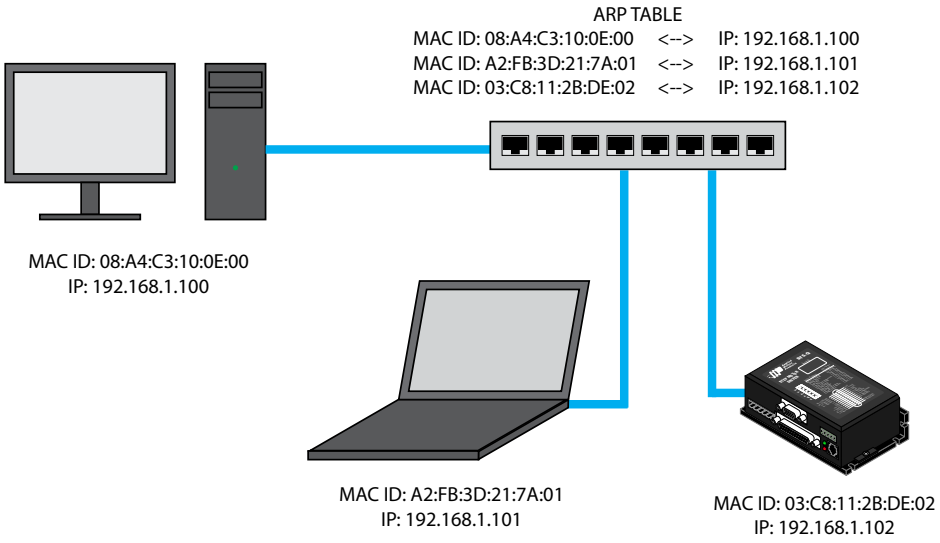
ARP Tables - The Ghost in the Machine

ARP, which stands for “Address Resolution Protocol”, is a low-level router function that enables traffic to be correctly routed on the Ethernet network. It is handled automatically by the router and is normally transparent to the user.

All network devices need to have two things: a MAC ID and an IP address.

- The MAC ID is a unique identifier that is assigned to the chip on the network interface device. You can think of it as a network serial number.
- The IP address is just that - an address. Like a street address on your house. IP addresses can be changed - MAC ID’s cannot.

The following diagram shows a basic network. Note that each device has both a MAC ID and IP address. The router maintains an ARP table, which is really just a list that matches MAC ID’s to IP addresses. An entry is created for every device on the network.



It should be noted that computers maintain a local ARP table as well, tracking other devices they’ve interacted with. This is an important point because the ARP table on a PC will typically refresh more frequently than those on a network router or switch.

So why do we care? Your application will probably require changing the IP address of a drive. The ARP table must then be refreshed to show the same MAC ID with a different IP address. This is usually not an issue if the drive is directly connected to the PC used to configure it, because the local ARP table will likely refresh quickly enough to catch the new IP address and re-establish a connection.

The problem comes when the drive is connected through a router during configuration. In this scenario it is entirely possible for IP address changes to happen more frequently than the ARP table can refresh itself. Most routers do not allow users to refresh the ARP table directly, as this poses a significant network security risk. The router must actually be rebooted to force a reset of the ARP table and allow a connection with the new IP address. Obviously this is not an ideal solution.

For this reason we recommend that all configuration be performed while directly connected to a PC. Do not use a router for drive configuration. Once an IP address is assigned the drive may be placed on the plant network without worry.

NOTE: If you find that you are changing IP addresses often and the connection becomes unreliable, it may be necessary to force a refresh of your PC’s local ARP table. This can be accomplished by opening a command window and using the command `arp -d`. You must have administrator privileges on your PC to do this.

Configuring Your Drive

Three Windows programs are available from Applied Motion Products for use with our Ethernet drives. These programs are included on the CD that accompanies each drive and the most recent version is always available at www.applied-motion.com.

ST Configurator is used to configure your stepper drive and motor. It can also be used to change the selection of drive IP addresses. ST Configurator includes extensive built-in help screens and manuals.

Quick Tuner is used to configure and tune servo drives. The Quick Tuner Manual is automatically installed in the Applied Motion Products program menu when you install Quick Tuner.

Q Programmer will be needed if you want to embed programs in the non-volatile memory of your drive, either to run automatically at power up or to be triggered by commands sent from a host.

Creating Your Own Application

To create your own application, you will need to choose a programming language, learn how SCL commands and responses are encapsulated in UDP packets, and learn to use your programming language's interface to the network.

UDP Packet Format

eSCL is based on Applied Motion's Serial Command Language (SCL), an ASCII-based language with roots in RS-232 and RS-485 communication. eSCL drives support the full SCL and Q command sets, and utilize the speed and reliability of Ethernet. Commands and responses are encapsulated in the payload of User Datagram Protocol (UDP) packets, and are transmitted using standard Ethernet hardware and standard TCL/IP stacks.

Sending Commands to a Drive

An eSCL UDP packet consists of three parts, the header (binary 07), the SCL string (a sequence of ASCII encoded characters) and the SCL terminator (ASCII carriage return, 13)

header	SCL string	<cr>

Example: Sending "RV"

- SCL Header = 07 (two bytes)
- R = ASCII 82
- V = ASCII 86
- <cr> (ASCII carriage return) = 13

header		"RV"		<cr>
0	7	82	86	13

Receiving Responses from a Drive

A typical response to "RV" would be "RV=103<cr>" which would be formatted as

header		"RV=103"						<cr>
0	7	82	86	61	49	48	51	13

Example Programs

Both example programs are available for download at www.applied-motion.com/example_code. You should still read this section so that you understand the key elements of the code and what tradeoffs you may encounter.

Visual Basic 6

Even though VB6 is an older language, its refreshing simplicity makes it a compelling choice for quickly developing an Ethernet application.

To communicate over Ethernet from VB6, you'll need the Winsock control (MSWINSCK.OCX), which is included in the Professional and Enterprise editions of the language. To configure an instance of Winsock, you must specify the protocol as UDP, choose a local port number, and set the remote IP address and port number to match the drive. In the code example below, 7775 is the port of the drive. driveIPaddress is the IP address of the drive ("10.10.10.10" or "192.168.0.130" for example). 7777 is the port of the PC.

```
Winsock1.RemotePort = 7775
Winsock1.RemoteHost = driveIPaddress
Winsock1.Protocol = sckUDPProtocol
Winsock1.Bind 7777
// if port 7777 is in use by another application, you will get an error.
// that error should be trapped using the On Error statement
// and an alternate port should be chosen.
```

Sending "RV" command:

```
Dim myPacket(0 to 4) as Byte   ' declare a byte array just large enough

myPacket(0) = 0                ' first byte of SCL opcode
myPacket(1) = 7                ' second byte of SCL opcode
myPacket(2) = "R"              ' R
myPacket(3) = "V"              ' V
myPacket(4) = vbCR             ' carriage return
Winsock1.SendData myPacket
```

To receive a response, you will need to place some code in the `Winsock_DataArrival` event. This event is automatically declared as soon as you add a Winsock control to your form. The DataArrival event will automatically trigger each time a packet is received. The code below extracts the SCL response from the UDP payload and displays it in a message box.

```
Private Sub Winsock1_DataArrival(ByVal bytesTotal As Long)
  Dim udpData() As Byte, n As Integer
  Dim hexbyte As String, packetID As Long, SCLrx As String
  Winsock1.GetData udpData
  ' remotehost gets clobbered when packet rec'd,
  ' next line fixes it
  Winsock1.RemoteHost = Winsock1.RemoteHostIP
  ' first 16 bits of packet are the ID (opcode)
  If UBound(udpData) >= 1 Then
    packetID = 256 * udpData(0) + udpData(1)
    If packetID = 7 then  ' SCL response
      SCLrx = ""
      For n = 2 To UBound(udpData)
        SCLrx = SCLrx & Chr(udpData(n))
      Next n
      MsgBox SCLrx
    End If
  End If
End Sub
```

C#.NET

The .NET languages are Microsoft's modern, object oriented Windows application building tools and include robust Ethernet support. We present this example in C#.

Make sure your project includes this line, providing access to an Ethernet socket:

```
using System.Net.Sockets;
```

In your form header you must declare a UdpClient object and create an instance, which can be done in the same line. The local port number is included in the "new UdpClient" call. This is the port number that will be reserved on the PC for your application.

```
static UdpClient udpClient = new UdpClient(7777);
```

To open the connection, invoke the Connect method, specifying the drive's IP address and port number:

```
udpClient.Connect("192.168.0.130", 7775);
```

To send "RV" to the drive:

```
//create a string loaded with the SCL command
Byte[] SCLstring = Encoding.ASCII.GetBytes("RV");
// create a byte array that will be used for the actual
// transmission
Byte[] sendBytes = new Byte[SCLstring.Length + 3];
// insert opcode (07 is used for all SCL commands)
sendBytes[0] = 0;
sendBytes[1] = 7;
// copy string to the byte array
System.Array.Copy(SCLstring, 0, sendBytes, 2, SCLstring.Length);
// insert terminator
sendBytes[sendBytes.Length - 1] = 13; // CR
```

```
// send it to the drive
udpClient.Send(sendBytes, sendBytes.Length);
```

Getting responses back from the drive in C# is a more complicated than VB6. You have two choices: poll for a response or create a callback function that will provide a true receive event.

Polling is easier to code but less efficient because you must either sit in a loop waiting for an expected response or run a timer to periodically check for data coming in. Since the choice depends on your programming style and the requirements of your application, we preset both techniques.

Polling for an incoming packet

The same UdpClient object that you use to send packets can be used to retrieve incoming responses from the drive. The Available property will be greater than zero if a packet has been received. To retrieve a packet, assign the Receive property to a Byte array. You must create an IPEndPoint object in order to use the Receive property.

```
private void UDPpoll()
{
  // you can call this from a timer event or a loop
  if (udpClient.Available > 0) // is there a packet ready?
  {
        IPEndPoint RemoteIpEndPoint = new IPEndPoint(IPAddress.Any, 0);
        try
        {
              // Get the received packet.  Receive method blocks
              // until a message returns on this socket from a remote host,
              // so always check .Available to see if a packet is ready.
              Byte[] receiveBytes = udpClient.Receive(ref RemoteIpEndPoint);
              // strip opcode
              Byte[] SCLstring = new byte[receiveBytes.Length - 2];
              for (int i = 0; i < SCLstring.Length; i++)
                    SCLstring[i] = receiveBytes[i + 2];
              string returnData = Encoding.ASCII.GetString(SCLstring);
              AddToHistory(returnData);
        }
        catch (Exception ex)
        {
              // put your error handler here
              Console.WriteLine(ex.ToString());
        }
  }
}
```

Creating a receive event using a call back function

First, create a function to handle incoming packets. This function must contain two local objects: a UdpClient and an IPEndPoint. The call back function will be passed an IAsyncResult object that contains a reference to the UDP connection. The local IPEndPoint object is passed to the UDPClient's EndReceive property to retrieve the packet.

```
public void ReceiveCallback(IAsyncResult ar)
{
	int opcode;
	UdpClient u = (UdpClient)((UdpState)(ar.AsyncState)).u;
	IPEndPoint e = (IPEndPoint)((UdpState)(ar.AsyncState)).e;

	Byte[] receiveBytes = u.EndReceive(ar, ref e);
	// get opcode
	opcode = 256 * receiveBytes[0] + receiveBytes[1];
	if (opcode == 7) // SCL response
	{
		string receiveString = Encoding.ASCII.GetString(receiveBytes);
		Byte[] SCLstring = new Byte[receiveBytes.Length - 2];
		// remove opcode
		System.Array.Copy(receiveBytes, 2, SCLstring, 0, SCLstring.
Length);
		receiveString = Encoding.ASCII.GetString(SCLstring);
		AddToHistory(receiveString);
	}
	else if (opcode == 99) // ping response
	{
		MessageBox.Show("Ping!", "eSCL Utility", MessageBoxButtons.OK,
MessageBoxIcon.Information);
	}
}
```

The call back function will not be called unless it is "registered" with the UdpClient object using the BeginReceive method, as shown below. StartRecvCallback can be called from the Form Load event. It must also be re-registered each time it is called (this is to prevent recursion), which is most easily accomplished by making a call to StartRecvCallback each time you send a packet.

```
private void StartRecvCallback()
{
	UdpState s = new UdpState();
	s.e = new IPEndPoint(IPAddress.Any, 0);
	s.u = udpClient;
	udpClient.BeginReceive(new AsyncCallback(ReceiveCallback), s);
}
```

This example requires that you declare a class called UdpState as described below.

```
class UdpState
{
	public UdpClient u;
	public IPEndPoint e;
}
```

As if this event driven technique wasn't quirky enough, it also creates a threading error unless the following statement in included in the form load event

```
// this must be so for callbacks which operate in a different thread
CheckForIllegalCrossThreadCalls = false;
```

Further Reading

The following materials can be downloaded from www. Applied-motion.com.

- The eSCL Utility will help you get familiar with the SCL language.
- ST Configuration Ethernet is needed to configure the ST5-QE and ST10-QE step motor drives. This application also includes extensive help screens.
- QuickTuner is used to configure and tune SV7 servo drives. . Quick Tuner also includes extensive help screens.
- Visual Basic and C# example projects can be downloaded from the software page.

To learn more about networking using Ethernet, we recommend reading Sams *Teach Yourself TCP/IP in 24 Hours*, available from amazon.com and other fine booksellers.

Appendix H: EtherNet/IP

EtherNet/IP products, designated by the letters “IP” in the model number, provide access to Q and SCL functionality over EtherNet/IP networks. This appendix details which commands are available and how to encapsulate them into EtherNet/IP and CIP packets. It is assumed that the user has a working knowledge of EtherNet/IP as it relates to the controller being used, as this chapter will not explain general EtherNet/IP implementation details.

For Allen Bradley PLC users, Applied Motion Products has developed a set of Add-On Instructions (AOIs) and associated Application Note. This set of AOIs can control Applied Motion Products Step, StepSERVO, and Servo drives from Rockwell CompactLogix or ControlLogix PLCs using a combination of implicit and explicit messaging. A user guide, an example RSLogix (for v20 and up) program, and the unlocked AOI files are included in the zip file. The AOIs can be found here.

Applied Motion Products offers both Class 1 and Class 3 type connections, each of which are useful for specific tasks. Class 1 connections are useful for high bandwidth tasks such as monitoring specific functions of the drive, while Class 3 connections are used for sending targeted messages to directly control the drive. The latter is used to implement Explicit Messaging.

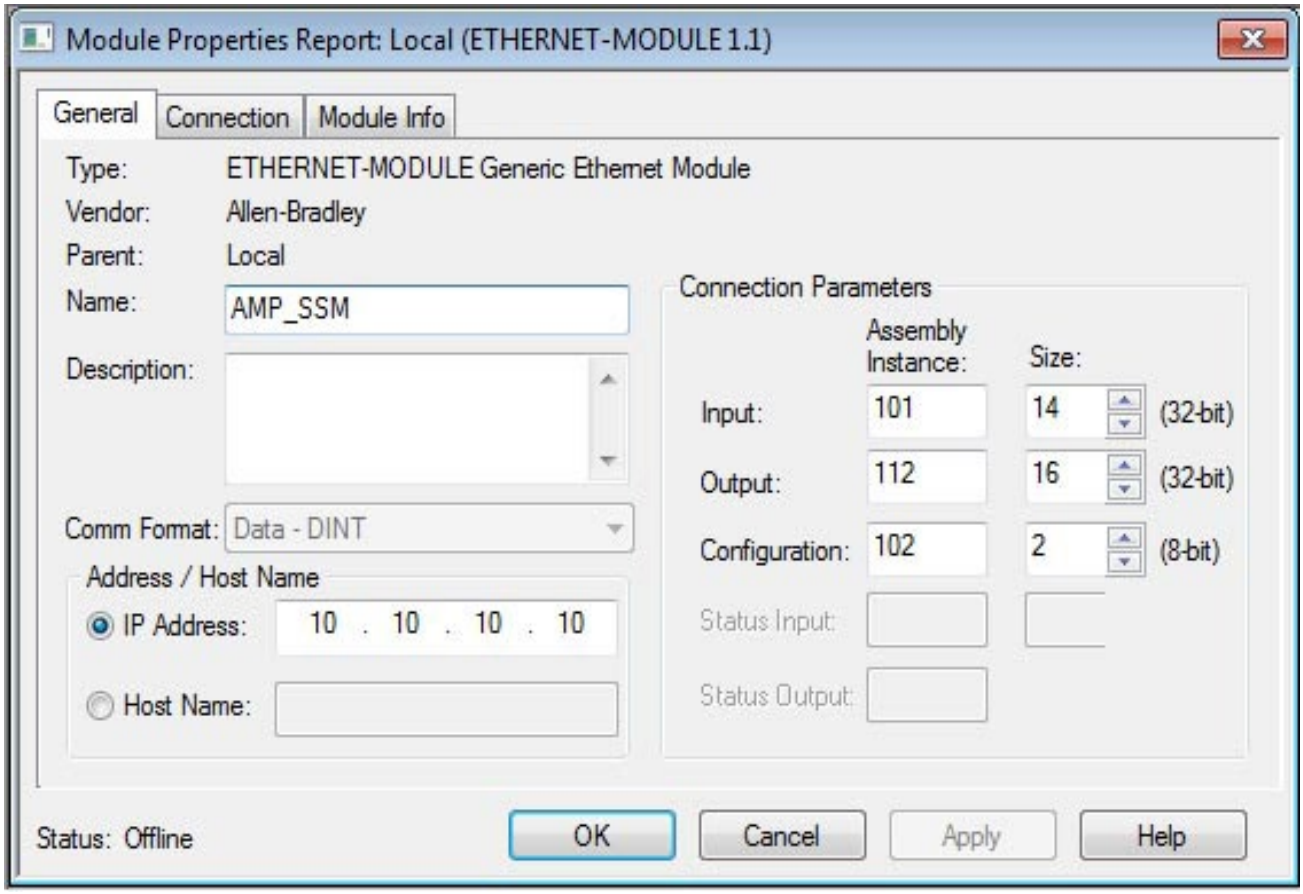
Note that with EtherNet/IP, all data direction notation assumes the point of view of the network. In this way, data sent by the drive to the controller is referred to as an Input, while data sent by the controller down to the drive is referred to as an Output.

The Class 1 Connection

Class 1 connections use Connection Points, which can be thought of as addresses with predefined functions. To communicate with an Applied Motion drive using a Class 1 connection, the following connection points are available.

Object ID		Size		Function	Notes
Hex	Decimal	32 bit DINTS	Bytes		
0x64	100	n/a	30	Input Assembly (Classic Version)	Static Assembly Object for monitoring drive status & behavior (see below for details)
0x65	101	14	56	Input Assembly (enhanced version)	Static Assembly Object (version 251.0 and later)
0x66	102	n/a	2	Configuration Assembly	Specifies parameters such as packet interval, data length.
0x67	103	0	0	Heartbeat Input Only Assembly	Zero-length message that tells the controller the drive is still active. Drive will multicast the specified input assembly (100 or 101)
0x68	104	0	0	Heartbeat Listen Only Assembly	Zero-length message that tells the drive the controller is still active.
0x70	112	16	64	Output Assembly	Static Assembly Object for controlling the drive (version 251 or later)

Typical PLC Connection Settings



Input Assembly (0x64)

This connection point is used to monitor the drive's behavior. The 30 bytes of data sent by the drive are as follows:

Field Descriptor	Length (bytes)
IP Address (Encoded in Internet Format)	4
Status Word	2
Alarm Word	2
Supply Voltage	2
Actual Current	2
Drive Temperature	2
Encoder Position (32-bit signed)	4
Absolute Position (32-bit signed)	4
Actual Velocity	2
Input Status (main board)	2
Input Status (extended)	2
Output Status	2

The data transmitted by the drive is sent in Little Endian format, so it will likely require rearranging before use.

IP addresses said to be stored in "Internet Format" are simply encoded into hexadecimal notation and rearranged into Little Endian format. Each octet has a value from 0-255, and can be represented by a single byte.

Standard IP address: *192.168.0.40*

Convert to Hexadecimal:
192 = 0xC0
168 = 0xA8
0 = 0x00
40 = 0x28

Rearrange into Little Endian Format: *C0 A8 00 28 -> 28 00 A8 C0*

Converted IP address: *192.168.0.40 -> 0x2800A8C0*

Note that all numbers are sent in Little Endian format, so the process for converting is the same for each piece of data.

Thus, an example message might be organized as follows:

Raw: E0032800A8C019000000E90100003802BAFCFFFFC72A0600C3FFFF40000F0F00

Grouped: [E003] [2800A8C0] [1900] [0000] [E901] [0000] [3802] [BAFCFFFF] [C72A0600] [C3FF] [FF40] [000F] [0F00]

The data should be decoded as follows. Where possible, the values have been converted to human-readable units. Please refer to the appropriate command page for further information. Note that Encoder Position, Absolute Position and Velocity are signed integers, and negative values will be represented in 2's complement form.

Sequence Number:	*0xE003*		
IP Address:	*0x2800A8C0*	*= 0xC0A80028*	*= 192.168.0.40*
Status (see SC command):	*0x1900*	*= 0x0019*	*= 0000 0000 0001 1001*
Alarm (see AL command):	*0x0000*		
Voltage:	*0xE901*	*= 0x01E9*	*= 489 (48.9 V)*
Current (see IC command):	*0x0000*		
Temp (see IT0 command):	*0x3802*	*= 0x238*	*= 568 (56.8 degrees C)*
Encoder Position (see EP command):	*0xBAFCFFFF*	*= 0xFFFFFCBA*	*= -838*
Absolute Position (see SP command):	*0xC72A0600*	*= 0x00062AC7*	*= 404167*
Velocity (see IV command):	*0XC3FF*	*= 0xFFC3*	*= -61*
Extended Inputs (see IS command):	*0xFF40*	*= 0x40FF*	
Main Board Inputs: (see ISX command):	*0x000F*	*= 0x0F00*	
Outputs (see IO command):	*0x0F00*	*= 0x000F*	

Input Assembly (0x65)

Available with firmware version 251.0 or later. This connection point is used to monitor the drive's behavior. The 14 DINTs of data sent by the drive are as follows:

Element	Field Descriptor	Length	Units	Notes
0	Status Code	32 Bits	see SC command	
1	Alarm Code	32 Bits	see AL command	
2	Supply Voltage	32 Bits	0.1 VDC	ex: 242 = 24.2 V
3	Actual Current (signed)	32 Bits	mA	ex: 1501 = 1.501A
4	Drive Temperature	32 Bits	0.1 deg C	ex: 305 = 30.5 deg C
5	Encoder Position (32-bit signed)	32 Bits	counts	20,000 counts = 1 shaft turn
6	Absolute Position (32-bit signed)	32 Bits	counts	20,000 counts = 1 shaft turn
7	Position Error (signed)	32 Bits	counts	
8	Actual Velocity (signed)	32 Bits	rev/sec * 240	ex: 480 = 2 rev/sec. This field reads zero if product does not have an encoder
9	Input Status (extended)	32 Bits	see below	also includes output status
10	Input Status (main board)	32 Bits	see below	also includes output status
11	(Reserved)	32 Bits		
12	Analog In 1	32 Bits	ADC counts	0 = min V, 16383 = max V
13	Analog In 2	32 Bits	ADC counts	0 = min V, 16383 = max V

Input Status Details

When inputs are closed, they read 0. When open they read according to bit code below. When outputs are open, they read 0. When outputs are closed, they read according to the code below. If more than one input is open (output is closed), the code will add up.

	I/O table - main board input status									main board outputs			
	bit code									bit code			
Product	IN1	IN2	IN3	IN4	IN5	IN6	IN7	IN8	enc index	OUT1	OUT2	OUT3	OUT4
SSM23IP	1	2	4	n/a	n/a	n/a	n/a	n/a	n/a	256	n/a	n/a	n/a
STM23/24IP	2	4	6	n/a	n/a	n/a	n/a	n/a	1	256	n/a	n/a	n/a
TSM34IP	1	2	4	8	16	32	64	128	n/a	256	512	1024	2048
ST5/10IP	1	2	4	8	16	32	64	128	16384	256	512	1024	2048
STF	1	2	4	8	16	32	64	128	n/a	256	512	1024	2048
TXM34IP	1	2	4	8	16	n/a	n/a	n/a	n/a	256	512	1024	n/a
TXM24IP	1	2	4	n/a	n/a	n/a	n/a	n/a	n/a	256	n/a	n/a	n/a
SWM24IP	2	4	8	n/a	n/a	n/a	n/a	n/a	1	256	n/a	n/a	n/a
STAC5IP*	2	4	8	16	n/a	n/a	n/a	n/a	1	256	512	n/a	n/a
*DB15 IN/OUT1 Connector													
	I/O table - extended input status									use 31..34 for extended outputs			
	bit code									bit code			
Product	IN1	IN2	IN3	IN4	IN5	IN6	IN7	IN8	enc index				
STAC5IP**	1	2	4	8	16	32	64	128	n/a	256	512	1024	2048
**DB25 OPT 2 Connector													

Bit Status Details - SV200

When inputs are closed, they read 0. When open they read according to bit code below. When outputs are open, they read 0. When outputs are closed, they read according to the code below. If more than one input is open (output is closed), the code will add up.

	bit code												bit code					
	IN1	IN2	IN3	IN4	IN5	IN6	IN7	IN8	IN9	IN10	IN11	IN12	OUT1	OUT2	OUT3	OUT4	OUT5	OUT6
SV200	1	2	4	8	16	32	64	128	256	512	1024	2048	4096	8192	16384	32768	65536	131072

Configuration Assembly (0x66)

This connection point is used by the EtherNet/IP protocol to configure various parameters including the Receive Packet Interval (RPI), data size, etc. It must be specified by the user.

Heartbeat Input Only Assembly (0x67)

This connection point represents a zero-length assembly object whose purpose is not to send data, but rather to simply inform the controller that the drive is still active and producing data.

Heartbeat Listen Only Assembly (0x68)

This connection point represents a zero-length assembly object whose purpose is not to send data, but rather to simply inform the drive that the controller is still active and receiving data.

Output Assembly (0x70)

Usage Available with firmware version 251.0 or later.

The first element in the Output Assembly is used to specify the type of command to be executed.

Assembly Element	Description	Length	Units	Notes
0	Command Word	32 Bits		
1	Jog Speed	32 Bits	Jog Speed 0.25 RPM See JS	See JS
2	Jog Accel	32 Bits	$10RPS^2$	See JA
3	Jog Decel	32 Bits	$10RPS^2$	See JL
4	Point-to-Point Velocity	32 Bits	0.25RPM	See VE
5	Point-to-Point Acceleration	32 Bits	$10RPS^2$	See AC
6	Point-to-Point Deceleration	32 Bits	$10RPS^2$	See DE
7	Point-to-point Distance	32 Bits	Motor Steps	See DI
8	Current Command (Servo and Step Servo Only	32 Bits	0.01A	
9	SCL Command Letters	32 Bits	2x ASCII characters only	
10	Data/SCL Register 1	32 Bits	Command dependent	
11	Data/SCL Register 2	32 Bits	Command dependent	
12	(Reserved)	32 Bits		
13	(Reserved)	32 Bits		
14	(Reserved)	32 Bits		
15	(Reserved)	32 Bits		

Assembly Element 0: "Command" Values:

Bit	Command Value	Description
0	0x0	Idle State (Send this between repeated commands)
	0x1	(reserved)
1	0x2	Enable
2	0x4	Disable
3	0x8	Start Move - FL
4	0x10	Start Move - FP
5	0x20	Start Move - FS (sensor)
6	0x40	Start Move - FD (double sensor)
7	0x80	Start Move - FY (sensor w/ safety distance)
8	0x100	Start Move - FM (sensor w/ mask distance)
9	0x200	Start Move - FO (set output)
10	0x400	Start Move - FC
11	0x800	Start Move - SH
12	0x1000	Start Move - FH
13	0x02000	Start Move - FE
14	0x4000	Stop/Kill - AM
15	0x8000	Stop/Kill - DE
16	0x10000	Start Jogging
17	0x20000	Update Jog Speed
18	0x40000	Send Host Command
19	0x80000	Q Load & Execute (QX)
20	0x100000	Alarm Reset - AR
21	0x200000	Current Command (GC) - servo and step servo only

This assembly be used to enable/disable the motor, execute velocity or position moves, or invoke the embedded SCL handler.

Note: With the exception of Send Host Command, these commands are edge triggered. Therefore, you must send an Idle State command between successive commands such as repeated moves and alarm resets.

Host Commands can be sent without an intervening Idle State if the command or associated parameter is changed.

Motor Enable

1. Assembly element 0: Transition from 0 -> 0x02 (bit 1). The motor will energize.
2. Assembly element 0: Reset to 0x0 (idle state) in preparation for the next command.

Motor Disable

1. Assembly element 0: Transition from 0 -> 0x04 (bit 2). The motor will disengage.
2. Assembly element 0: Reset to 0x0 (idle state) in preparation for the next command.

Point-to-Point Positioning Move (FL/FP)

To begin a move, the user must specify the appropriate move parameters, and then trigger the move.

1. Assembly element 4: Velocity (units: 0.25 RPM)
2. Assembly element 5: Accel (units: 10 rps^2)
3. Assembly element 6: Decel (units: 10 rps^2)
4. Assembly element 7: Distance
5. Assembly element 0: Transition from 0 -> 0x08 (bit 3) to begin an FL (incremental) move, or transition from 0 -> 0x10 (bit 4) to begin an FP (absolute) move.
6. Assembly element 0: Reset to 0x0 (idle state) in preparation for the next command.

Point-to-Point Positioning Move with I/O (FS/FD/FY/FS/FO/SH/FE)

1. Specify move speed in Assembly element 4 (units: rev/sec*240)
2. Specify accel rate in Assembly element 5 (units: rev/sec/sec*6)
3. Specify decel rate in Assembly element 6 (units: rev/sec/sec*6)
4. Specify distance in Assembly element 7 (units: motor steps)
5. Specify IO point and condition in Data/SCL Register 1 (see IO Encoding Table)
6. Specify 2nd IO point and condition in Data/SCL Register 2 (FD only)
7. Specify Start Move by setting Assembly element 0 to appropriate value from Command Word Table.
8. Set Assembly element 0 to 0 in preparation for the next command.

Stop/Kill (AM, “crash stop”)

1. Assembly element 0: Transition from 0 -> 0x4000 (bit 14). The motor will stop.
2. Assembly element 0: Reset to 0 (idle state) in preparation for the next command.

Stop/Kill (DE, “normal stop”)

1. Assembly element 0: Transition from 0 -> 0x8000 (bit 15). The motor will stop.
2. Assembly element 0: Reset to 0x0 (idle state) in preparation for the next command.

Velocity (Jog) Move

1. Specify jog speed in Assembly element 1 (units: rev/sec*240) (To jog in CCW direction, enter a negative speed)
2. Specify accel rate in Assembly element 2 (units: rev/sec/sec*6)
3. Specify decel rate in Assembly element 3 (units: rev/sec/sec*6)
4. Specify Start Jogging by setting Assembly element 0 to 0x10000 (bit 16).
5. Set Assembly element 0 to 0 in preparation for next command.

Update Jog Speed (change speed while in motion)

1. Assembly element 1: new jog speed (units: 0.25 RPM) (To jog in CCW direction, enter a negative speed)
2. Assembly element 0: Transition from 0x20000 (bit 17) to update jog speed in real time.
3. Assembly element 0: Reset to 0x0 (idle state) in preparation for the next command.

Note: jog accel and jog decel cannot be changed while the motor is moving.

Embedded SCL Handler

(see table on following pages for command list)

General usage is as follows:

1. Assembly element 9: SCL command characters
    a. NOTE: Always two ASCII characters, in the lower 16 bits
2. Assembly element 10 (Data/SCL Register 1)
3. Assembly element 11 (Data/SCL Register 2)
4. Assembly element 0: Transition from 0x40000 (bit 18) to execute the command.
5. Assembly element 0: Reset to 0x0 (idle state) in preparation for the next command.

Q Load and Execute (QX)

1. Assembly element 10: Segment to execute
2. Assembly element 0: Transition from 0 -> 0x80000 (bit 19) to begin executing the specified Q segment.
3. Assembly element 0: Reset to 0x0 (idle state) in preparation for the next command.

Point-to-Point Move with Speed Change (FC)

1. To specify the distance at which you want the motor speed to change, use the Host Command DC.
2. To specify the speed you want the motor to change to, use the Host Command VC.
3. Specify the initial move speed in Assembly element 4 (units: rev/sec*240)
4. Specify accel rate in Assembly element 5 (units: rev/sec/sec*6)
5. Specify decel rate in Assembly element 6 (units: rev/sec/sec*6)
6. Specify distance in Assembly element 7 (units: motor steps)
7. Specify Start Move by setting Assembly element 0 to 0x400 (bit 10).
8. Set Assembly element 0 to 0 in preparation for the next command.

Alarm Reset

1. Request the alarm reset by setting assembly element 0 to 0x100000 (bit 20).
2. Set Assembly element 0 to 0 in preparation for the next command.

For commands involving inputs and outputs, such as FO, FS and SO, you'll need to specify the IO and condition in Data/SCL Register 1.

Example 1: SO2L

Set the SCL Command Letters field of the output assembly to 'SO'. If your programming environment does not support ascii strings, use the hex equivalent of 'SO': 0x534F.
Set the Data/SCL Register1 field to the hex code for 2L, from the table below: 0x4CB2
Set the command word bit for Host Command: bit 18 (0x40000)

Example 2: FS3R

Set the Data/SCL Register1 field to the hex code for 3R, from the table below: 0x52B3
Set the command word bit for Feed to Sensor: bit 5 (0x20)

Data/SCL Register codes for IO commands:

	Enc Index	IN1	IN2	IN3	IN4	IN5	IN6	IN7	IN8	IN9	IN10	IN11	IN12
High	0x48B0	0x48B1	0x48B2	0x48B3	0x48B4	0x48B5	0x48B6	0x48B7	0x48B8	0x48B9	0x48BA	0x48BB	0x48BC
Low	0x4CB0	0x4CB1	0x4CB2	0x4CB3	0x4CB4	0x4CB5	0x4CB6	0x4CB7	0x4CB8	0x4CB9	0x4CBA	0x4CBB	0x4CBC
Rising Edge	0x52B0	0x52B1	0x52B2	0x52B3	0x52B4	0x52B5	0x52B6	0x52B7	0x52B8	0x52B9	0x52BA	0x52BB	0x52BC
Falling Edge	0x46B0	0x46B1	0x46B2	0x46B3	0x46B4	0x46B5	0x46B6	0x46B7	0x46B8	0x46B9	0x46BA	0x46BB	0x46BC