EZ Servo®
Command set and Communications protocol

Command Set For Servo Models: EZSV23, EZSV17

Document Revision: A51 9/1/14

INDEX

EZ Servo®
Command set and Communications protocol

Overview:
This document describes the operation and command set for the EZServo® series of motor drives.

Command syntax:
Commands to the EZServo are single alpha characters normally followed by a numeric value. The alpha character represents "what to do" and the numeric value represents "how much to do it".
You can set values for desired velocities, accelerations, and positions. Commands can be issued one at a time or sent in a group. This allows the setting of all move parameters in one command. You can also create loops in the strings and cause the EZ Stepper or EZServo to become a stand-alone device that responds to switch inputs. Finally, storing such strings into the onboard EEPROM allows the EZServo to power up into a mode of your choice, so that it can act with no computer attached.
The Commands are simply typed into a Terminal Program such as "Hyperterminal", no special software is required. The EZServo can even be commanded from a serial enabled Tablet Computer.

EZ Servo®
Command set and Communications protocol

Command Set:
(Also see examples on page 17)

Command (**Case sensitive**)	Operand	Description
		POSITIONING COMMANDS
A	0-(2^31)	**Move Motor to absolute (** units are quadrature encoder ticks- 32 Bit Positioning**). Eg /1A1000R** (Note: for a finite move, the ending Absolute position must be greater than between 0 and +2x10^8 for V5.45 firmware or between +/-2x10^8 for V7+ firmware).
P	0- (+/- 2x10^8)	**Move Motor relative in positive direction.** **(** units are quadrature encoder ticks**) Eg/1P1000R** A value of zero will cause an endless forwards move at speed V. (i.e. enter into Velocity Mode) The Velocity can then be changed on the fly by using the "V" command. An endless move can be terminated by issuing a T (Note: for a finite move, the ending Absolute position must be greater than between 0 and +2x10^8 for V5.45 firmware or between +/-2x10^8 for V7+ firmware).
D	0- (2x10^8)	**Move Motor relative in negative direction.** **(** units are quadrature encoder ticks**)** (Note: for a finite move, the ending Absolute position must be greater than between 0 and +2x10^8 for V5.45 firmware or between +/-2x10^8 for V7+ firmware). A value of zero for the operand will cause an endless backwards move at speed V. (i.e. enter into Velocity Mode). The Velocity can then be changed on the fly by using the V command. An endless move can be terminated by issuing a T
B	0-3390	Sets Distance moved in "Bump Jog Mode" see "n" command. This jog mode is enabled by /1n1R. Jog by this amount of distance occurs when the switch inputs are connected to ground.

		HOMING COMMANDS
Z	**0-(2^31)** (400)	**Home/Initialize Motor.** Motor will turn toward 0 until the home opto sensor (opto #1) is interrupted. If already interrupted it will back out of the opto and come back in until re-interrupted. Current motor position is set to zero. See Appendix 2 for further details. **Eg /1Z300000R**
z	**0-(2^31)**	**Change current position without moving.** Sets both Encoder and Commanded position to value given, without moving the motor. **Eg /1z1000R** Note that this command must be issued after the servo is on, if it needs to be the first command then issue an A0 first eg /1A0z1000R .
r	**0 or 1** (0)	**Recover Servo, Turn off Servo** Turns the Servo on or off to current encoder position. This is useful in the case where an overload error has occurred and the servo has automatically shut down, but the encoder count is still valid. Operation may be resumed without re-homing. Also see /1n8R command. Typically this command would be used in conjunction with the n512 mode in a recovery script in case the servo detects an overload and times out. Also /1r0R will turn off the servo so the motor can be turned by hand, and a subsequent /1r1R command will turn on the servo to the new position.
f	**0 or 1** (0)	**Home Flag and Limit polarity.** Sets polarity of home sensor, default value is 0. **Eg /1f1R** When f=0 limits are activated when limit inputs are high. When f=1 limits are activated when limit inputs are low. Starting in V7.08 the "ap" command can also change polarity.

		SET VELOCITY COMMANDS
V	**1- (2^23)** (1,000,000)	**In Position Mode (N1) Set Max/Slew Speed of motor** For Servo version with an encoder, sets (encoder ticks/32.768) per second. (Eg Value of 33 will cause the motor to spin at approx 1 encoder tick/second)
V	**1-1000** (0)	**In Velocity Mode (N0) Set Spin Speed Of Motor** For Servo BLDC Version with only Hall Sensor Feedback (i.e. no encoder). Sets speed in Revs/Second. Use /1N0P0R or /1N0D0R Command to get started, and set direction. Then issue /1V100R etc
U	**1- (2^23)** (1,000,000**)**	**Same as V command for Position Mode except sets velocity in negative direction.** This must be set after setting V, any subsequent setting of V will set both U and V to be the same.
		SET ACCELERATION COMMAND
L	**0-65000** (4000)	Sets Acceleration and Decelerstaion Acceleration in encoder ticks/second squared = = ((L Value) x (4000000)) / (65536) Eg with V=1000000 and L=10 and "A" set very large at 10^8. Issue command /1V1000000L10A10000000R Wait for motor to reach a steady speed. Type /1T and measure the time to decelerate to a stop. Time to decelerate = V/L = (1000000/32.768) / ((10*4000000)/65536) = 50 seconds
aL	**0-65000**	Sets Only Deceleration if command issued after acceleration, eg /1L100aL1000R Note: any issue of a subsequent L command overwrites this value and sets Accel = Decel , aL = L
aaL	**0-65000**	Sets Only Deceleration when a Limit is hit, if command issued after acceleration, eg /1L100aaL10000R Note: any issue of a subsequent L command overwrites this value and sets aaL = L

		LOOPING AND BRANCHING COMMANDS
g		**Beginning of a repeat loop** marker. See examples below on how to set up a loop.
G	0-30000	**End of a repeat loop** marker. Loops can be nested up to 4 levels. A value of 0 causes the loop to be infinite. (Requires T command to Terminate). If no value is specified 0 is assumed. **Eg /1gP1000M1000G10R**
M	0-29000	Wait for this number of milliseconds Eg. **/1gA0M1000A1000M1000G0R** will go back and forth between 0 and 1000 and wait 1 second at each position. (Accurate to +/-10mS)
H	 01 11 02 12 03 13 04 14 07 17	**Halt** current command string and wait until condition specified. Wait for low on input 1 (Switch 1) Wait for high on input 1 (Switch 1) Wait for low on input 2 (Switch 2) Wait for high on input 2 (Switch 2) Wait for low on input 3 (Opto 1) Wait for high on input 3 (Opto 1) Wait for low on input 4 (Opto 2) Wait for high on input 4 (Opto 2) Wait for low on Index of encoder V7.08+ Wait for high on on Index of encoder V7.08+ If Halted operation can also be resumed by typing /1R Also see "S" command for I/O dependant program execution. If an edge detect is desired, a look for Low and a look for Hi can be placed adjacent to each other eg H01H11 is a rising edge detect. **Eg /1gH12H02P1000G0R** will advance the motor 1000 steps every time the switch #2 is pressed.

S	 01 11 02 12 03 13 04 14 07 17	**Skip next instruction depending on status of switch.** Skip next instruction if low on input 1 (Switch 1) Skip next instruction if high on input 1 (Switch 1) Skip next instruction if low on input 2 (Switch 2) Skip next instruction if high on input 2 (Switch 2) Skip next instruction if low on input 3 (Opto 1) Skip next instruction if high on input 3 (Opto 1) Skip next instruction if low on input 4 (Opto 2) Skip next instruction if high on input 4 (Opto 2) Skip next instruction if low on Index of encoder V7.08+ Skip next instruction if high on Index of encoder V7.08+ Program branching to a complex subroutine can be implemented by making the next instruction a stored string execution. (See examples). Loops can be escaped by branching to a stored string with no commands. Also see "H" command for I/O dependant program execution.
ap	1 2 4 8	**Invert the polarity of each input (V7.08+)** This command will invert each of the polarities of each of the inputs. The inversion will show in /1?4, f,S, H, and Z commands. Example **/1ap10R** will invert input 2 and 4. (10 = 2+8). The ap command should not be used immediately prior to command that depends on it. Eg /1ap15H02 may not always execute correctly.
d	1-30000	**Debounce** -Number of times the binary pattern on the 2 switches must be constant for prior to acting upon the "i." command.
i	0 1 2 3	**Skip on status of 2 switches.** The "i" command interprets 2 switches as a combined binary number, with values 0,1,2,3 The next instruction is skipped unless the number is what is read back. Eg /1d1000gi0A0i1A1000i2A2000i3A3000G0R It is possible to implement a debounce on this command in case the switches do not simultaneously change by use of the "d" or debounce command. Allowed values of "d" are 1-30000.
d	1-30000	**Debounce** -Number of times the binary pattern on the 2 switches must be constant for prior to acting upon the "i." command.

		PROGRAM STORAGE AND RECALL
s	0-15	**Stores a program** 0-3 or 0-15 depending on model, Program 0 is executed on power up. (25 full commands max per string). Note: This command takes approx 1 Second to write to the EEPROM. Note: The first stored command in a string cannot start with a lower case "a"
e	0-15	**Executes Stored Program** 0-15.
		PROGRAM EXECUTION
R		**Run** the command string that is currently in the execution buffer.
X		**Repeat Run** the current command string
		SET MAX MOVE CURRENT / PID CONSTANTS / TORQUE MODE / HOLD CURRENT
m	**0-100** (50)	**Sets Max current allowed during a move.** 100% = 5A for EZSV23 . (Default is 50%) 100% = 2A for EZSV17 . (Default is 50%)
h	**0-50** (0)	**Sets Current in Torque mode.** 100% = 5A for EZSV23 . (Default is 0) 100% = 2A for EZSV17 . (Default is 0) When a value is written in here the servo automatically enters Torque mode. Issue P0 or D0 to change direction. Starting in V7.08 this command will go to 100% /1h100R
ah	**0-1000** (0)	Sets PWM motor Voltage in the case of Brush Motors for velocity mode with no feedback. See Appendix 5
u	**1-25000** (25000)	**Overload Timeout (Milliseconds)** When the Servo detects an overload condition it will shut down after this number of Milliseconds. Default 25000. See appendix 8 for recovery from an overload .
au	**1-65000** (100)	**Overload Retry Count** When the n modes 512 and 1024 are activated to retry a stalled move, then that move must complete before this number of retries. After this number of retries is exhausted then stored program 12 is executed. See appendix 8 for recovery from an overload .
ar	**5073**	**Processor Reset** This command will initiate a Processor Reset. 5073 is chosen to avoid inadvertent resets. Eg: /1ar5073R<CR> Use this only in an emergency, because the loads will be uncontrolled while the processor reboots and the PTC fuse may trip, and a power-cycle may be required.
w	**0-65534**	**Set Servo Proportional gain.**

	(1000)	(Default value is 1000, and is stable with most motors, when equipped with 400-1000 line encoders – 1600-4000 quadrature ticks). Default is 1000. See Appendix 6
x	**0-65534** (0)	**Set Servo Set Integral gain.** (Default value is 0, and is stable with most motors, when equipped with 400-1000 line encoders – 1600-4000 quadrature ticks). Default is 0. See Appendix 6
y	**0-1365534** (2500)	**Set Servo Set Differential gain.** (Default value is 2500, and is stable with most motors, when equipped with 400-100 line encoders – 1600-4000 quadrature ticks). Default is 2500. See Appendix 6
		SPECIAL MODES COMMANDS
j	**2, 4, 8, 16, 32, 64, 128, 256** (256)	Reserved (Future Release) Pulses Received on the Step and direction inputs are multiplied by this number and divided by 256. The Resulting number is then added/ subtracted from the current position measured in encoder ticks.
N	**0-3** (1)	**Sets Modes – Interpret as combination of Binary Bits** 0 = No Encoder – Uses Hall Sensors for Velocity Control. 1 = Encoder With No Index (Default). Homes to Opto. 2 = Encoder With Index. Homes to Index. Use a slow velocity for homing in this mode (1 rev/sec). See Appendix 2. Note: Hall sensors can be wired as an encoder for crude position control. (Must Be 5Volt). See Appendix 5 3 = Uses Potentiometer 1 as an encoder. See Appendix 9. 4= Reserved 5 = CANBUS Slave mode. In this mode the unit is a slave to position messages received via the CANBUS EZLink Connector. (future release) 6 = Reserved 7 = High Speed Index homing, requires hardware modification, contact factory. Firmware V7.08+ 8 = Uses Channel A of encoder connector for as a tachometer feedback for velocity mode. This is 2 quadrant mode that is not as good at a full encoder feedback 4 quadrant mode.
n	**0-65535** (0)	**Sets Modes – Interpret as combination of Binary Bits** Note: These bits can be combined eg: issue /1n6R for 2+4 **Bit0 (LSB)** : /1n1R Enable Pulse Jog Mode. Jog distance is given by “B” command. Velocity is given by “V” command. The Switch Inputs become the Jog Inputs. Note that the Jog Mode will need to be turned off for sending

		regular positioning commands A P D etc. Also note that the servo needs to be on /1A0n1R etc. **Bit1 :** /1n2R Enable Limits. (The two optos become limits switches). The polarity of the limits is set by the "f" command. The Home opto is the lower limit. **Bit2 :** /1n4R Enable Continuous Jog Mode. Continuous run of motor while switch is depressed. Velocity is given by "V" command. Note that in software less than V7 which did not have negative numbers, the jog mode allows moves below zero, which will be interpreted by any subsequent "A" command as a large positive number. If this is undesirable, please use the "z" command to define zero position to be some positive number so that underflow will not occur. Note that the Jog Mode will need to be turned off for sending regular positioning commands A P D etc. Also note that the servo needs to be on /1A0n1R etc. **Bit3 :** /1n8R When Set the encoder value will be preserved when an overload shutdown occurs. Any subsequent "A" command after an overload will restart the servo while preserving the encoder count. (Requires V 7.X or later software). Also see /1r1R command. **Bit4 :** /1n16R Reserved **Bit5 :** /1n32R Enable Step And Direction Mode if (1) or enable Dual Encoder Mode if (0) **Eg /1n96<CR>** (96=32+64) Enables step and dir mode and slaves the motor to it. Step input is Switch 2 and direction is Switch 1 input. See wiring Diagram for encoder hookup. **Bit6 :** / 1n64R Enable Motor slave to encoder/step-dir. **Bit7 :** /1n128R Reserved **Bit8 :** /1n256R When Set this bit will disable the response from the servo. **Bit9 and Bit10:** When set these bit will execute one of the stored programs 13, 14 or 15 whenever the servo shuts down due to an overload or an error. /1n512R will execute program 13 /1n1024R will execute program 14 /1n1536R will execute program 15 See appendix 8 for an example. **Bit11** /1n2048R Reserved. **<u>Modes below require V6.X + Code</u>** **Bit12 :** /1n4096R CANBUS master mode (SV23 Only). This mode will send out CANBUS messages on the CANBUS auxiliary connector. These messages can be used to electronically gear several SV23 drives together.

		Bit13: /1n8192R Uses Potentiometer 2 to command the motion of the motor. See Appendix 9. **Bit14:** /1n16384R When Set this bit will kill any move if switch 1 is pushed. **Bit15:** /1n32768R When Set this bit will kill any move if switch 2 is pushed. **Bit16:** /1n65536R When Set Potentiometer on Opto 2 input will set velocity. (Joystick Mode) See Appendix 9.
an	**0-65535** (0)	**Sets Extended Modes – Interpret as combination of Binary Bits** Note:These bits can be combined eg: issue /1an6R for 2+4 **Bit0 (LSB) :**/1n1R Reserved **Bit1 :** /1an2R Turns on software limits (set by aaP aaD) **Bit2 :** /1an4R Reserved **Bit3 :** /1an8R Reserved **Bit4 :** /1an16R Turns on Auto-Restart of P0 Mode when in potentiometer velocity (Joystick) mode. **Bit5 :** /1an32R Reserved **Bit6 :** /1an64R Reserved **Bit7 :** /1an128R Reserved **Bit8 :** /1an256R Reserved **Bit9 :** /1an512R Reserved **Bit10 :** /1an1024R Reserved **Bit11 :** /1an2048R Reserved **Bit12 :** /1an4096R Reserved **Bit13 :** /1an8192R Reserved **Bit14 :** /1an16384R When Set will use switches for limits. **Bit15 :** /1an32768R Reserved
b	**9600** **19200** **38400** **to 230400** (9600)	**Adjustable baud rate** **Eg /1b19200R** This command will usually be stored as program zero and execute on power up. Default baud rate is 9600.
p	**0-65000**	**Ping Command (lower case "p")** This command will send a numeric message back to the host, when that point in the string is reached. **Eg /1gA1000p3333A0G0R** Will send the number 3333 every time through the loop. Care must be taken when using this command because it can tie up he 485 bus.

		ON/OFF POWER DRIVER
J	**0-3** (0)	On/Off Driver – Interpret as 2 bit Binary Value, 3=11= Both Drivers On, 2=10=Driver2 on Driver 1 Off etc. Note: Only the EZSV23 Has the On/Off Drivers wired to Pins . Starting V7.08 these pins will also activate on the micro-processor on the SV17 and can be jumpered to output connectors if desired.
aK	**0-500**	PWM Output command (SV23 with V7.0+ firmware) This command drives ON/OFF Driver #1 in PWM Mode./1aK250R The frequency is 20KHz, The value is the ON time in increments of 50uS. Setting the value to Zero returns the driver to J command mode. (Drivers Must be derated to 0.5A in this mode, due to switching losses) Only the EZSV23 Has the On/Off Drivers wired to Pins . Starting V7.08 these pins will also activate on the micro-processor on the SV17 and can be jumpered to output connectors if desired.
aaK	**0-500**	Same as aK command but for Driver #2
		DEVICE RESPONSE PACKET
		See Appendix 7 for detailed description of device response to commands.

		ANALOG TO DIGITAL CONVRTR COMMANDS
		All 4 Inputs are ADC inputs and can be read and acted upon by the program. Please see Appendix 9
at	**100000-416368**	Thresholds where the Drive calls a one or a zero for each input is varied by this command.
?aa		Reads back all 4 Input ADC Values **E.g. /1?aa<CR>** The Readback order is channels 4:3:2:1
at	**100000 to 116368** **200000 to 216368** **300000 to 316368** **400000 to 416368** (6144)	The “at” command sets the threshold, upon which a “one” or “zero” is called for each of the 4 channels. The Number represents the channel number followed by a 5 digit number from 00000-16368 which represents the threshold on a scale from a 0-3.3V. The default values are 6144 for all 4 channels which represents 1.24V. Changing the threshold allows the H and S commands to work on a variable analog input value which essentially allows the program to act upon an analog level. **Eg /1at106144R** sets the threshold of channel 1 to 6144, Note that leading zeros are required for the threshold value which is always 5 digits plus the channel number.
?at		Reads back the thresholds for all 4 channels. The Readback order is channels 4:3:2:1 **Eg /1?at<CR>**
		POTENTIOMETER POSITION COMMAND
		The motion of the Servo can be slaved to value read from Potentiometer2. Please see appendix 9.
ao	**0-20000** (0)	After multiplication by the am value this offset is added to obtain the position command. **Eg /1ao1000R<CR>**
am	**0-20000** (256)	The Pot value is multiplied by this value and divided by 256 to get the position command. **Eg /1am512R<CR>**
ad	**0-20000** (50)	Sets a deadband around the pot value used for the last move, which must be exceeded before a new move command is issued. **Eg /1ad100R<CR>**

		POTENTIOMETER VELOCITY COMMAND
		(Joystick Mode) - Future Command not yet available - The velocity of the Servo can be slaved to value read from Potentiometer2. Please see appendix 9. Typically type /1n65536zP0R to enter this mode
am	**0-120000** (256)	The Pot value is multiplied by this value and divided by 256 to get the position command. **Eg /1am512R<CR>**
ad	**0-20000** (50)	Sets a deadband around the mid pot value, which must be exceeded before movement will start **Eg /1ad100R<CR>**
z	**0-16384** (current pot value)	Sets pot value for zero velocity. i.e. midpoint of joystick. **Eg /1z8000R<CR>** If zero value is specified, the current value of the pot at the time of issuing this command will be used as the midpoint of the pot. **Eg /1z0R**
		MISC COMMANDS
aP	**5**	Sets delay in milliseconds in between a command being received and the response from the driver being sent out on the 485 bus.
aK	**0-1000** (0)	For Piezo Servo only. Set Minimum amplifier power value. This is useful for compensating for deadbands / stiction in motors. Used for Piezo Drives.

<u>Hardware protocol:</u>
The EZ Servo communicates over the RS485 bus at 9600 baud, 1 Stop bit, No Parity, No flow control.

<u>Commands Below are "Immediate" Commands, and cannot be cascaded in strings or stored. These commands execute while others commands are running.</u>

		IMMEDIATE QUERY COMMANDS
T		Terminate current command or loop. **(Eg: /1T <CR>)**
?	0 A	Returns the current Commanded motor position **/1?0<CR>** Or in V7.08+ **/1?A** also works
?	1	Reserved
?	2 V aV	/1?2Returns the current Slew/Max Speed for Position mode. /1?V returns the programmed speed, (this is the accn commanded by the trajectory generator and will change during accel and decel ramps) /1?aV returns the actual speed based on encoder count subtraction. Requires V7.06+ software. This value can be niosey.
?	3	Reserved
?	4	Returns the status of all four inputs, 0-15 representing a 4 bit binary pattern. **Eg /1?4<CR>** Bit 0 = Switch1 , Bit 1 = Switch2 Bit 2 = Opto 1, Bit 3 = Opto 2
?	a4	**Returns the status of all four inputs, plus the value of the primary encoder inputs, read as digital I/O bits 0-127, representing a 7-bit binary pattern. /1?a4.** Requires V7.08+ firmware Bit 0 = Switch1 Bit 1 = Switch2 Bit 2 = Opto 1 Bit 3 = Opto 2 Bit 4 = Encoder CHA Bit 5 = Encoder CHB Bit 7 = Encoder Index
?	5 , 6, 7	Reserved
?	8	Returns Encoder Position. (can be zeroed by "z" command)
?	9	Erases all stored commands in EEPROM.
?	10	Returns Second Encoder or Step and Direction Counter Readback . Requires V7.X firmware.
?	w	Returns Proportional Gain Value **Eg /1?w<CR>**
?	x	Returns Integral Gain Value **Eg /1?x<CR>**
?	y	Returns Differential Gain Value **Eg /1?y<CR>**
&		Returns the current Firmware revision and date **/1&<CR>**
Q		Query current status of EZServo Returns the Ready/Busy status as well as any error conditions in the "Status" byte of the return string. **The Return string consists of the start character (/), the master address (0) and the status byte.** Bit 5 of the status

		byte is set when the EZServo is ready to accept commands. It is cleared when the EZServo is busy. The least significant four bits of the Status byte contain the completion code. The list of the codes is: 0 = No Error, 1 = Initialization error 2 = Bad Command, 3 = Operand out of range Errors in OpCode will be returned immediately, while Errors in Operand range will be returned only when the next command is issued.
$		/1$<CR> Reads back the last command strings that was run.
		Some commands are new and present only in later models. .

Return message decode procedure:

The bus is bi-directional and there may be spurious characters generated during the bus turnaround. Therefore decoding of the return message should not be done by counting characters after the end of the transmission, instead should be done by parsing for “message to master” which is /0.

Examples of a command strings in DT protocol are:

Please first see Appendix 5 and 6 to ensure correct wiring and stability of motor, and also download and install the EZCommander application from the support page.

Example #1 (A Move to Absolute Position)

/1A12345A55R<CR>

This breaks down to:

1. "/" is the start character. It lets the EZ Servos know that a command is coming in.
2. "**1**" is the device address, (set on address switch on device).
3. "**A12345**" makes the motor turn to **A**bsolute position **12345**
4. "**A55**" makes the motor turn to **A**bsolute position **55**
5. "**R**" Tells the EZ Servo to **R**un the command that it has received.

<CR> is a carriage return (ie hit enter button) that tells the EZ Servo that the command string is complete and should be parsed.

Noteing Hyperterminal: Hyperterminal issues each character as you type it in. Therefore it is not possible to cut and paste in Hyperterminal.

Example #2 (Move loop with waits)

/1gA10000M500A0M500G10R<CR>

This breaks down to:

1. "/" is the start character. It lets the EZ Servos know that a command is coming in.
2. "**1**" is the device address, (set on address switch on device).
3. "**g**" is the start of a repeat loop
4. "**A10000**" makes the motor turn to **A**bsolute position **10000**
5. "**M500**" causes the EZ Servo to wait for **500 M**illiseconds.
6. "**A0**" makes the motor turn to **A**bsolute position **0**.
7. "**M500**" is another wait command for **500 M**illiseconds.
8. "**G10**" will make the string repeat 10 times starting from the location of the small "g"
9. "**R**" Tells the EZ Servo to **R**un the command that it has received.
10. **<CR>** is a carriage return that tells the EZ Servo that the command string is complete and should be parsed.

Note that a multi command string is executed in sequence from left to right.
To Terminate the above loop while in progress type **/1T** . Any loop or move in progress must be terminated prior to sending a new command or string of commands.

Example #3 (Program Storage and Recall)

An example of a storing a command string for later execution:

/1s2gA10000M500A0M500G10R<CR>

The program outlined in the prior example is stored as Program 2

/1e2R<CR>

Will execute the previously stored program #2. (Note: program 0 is always executed on power up, if we use 0 instead of 2 in the above example then this program would execute automatically on power up). Note that the max limit on a string length is 22 commands or 256 characters, which ever is less. However Strings can be run from each other by use of the "e" commands giving 22 x 16 max contiguous commands.
Note that it takes about 1 second to store a program, (the strings read back in 10mS)
Note: The first stored command in a string cannot start with a lower case "a"

Example #4 (Set Current , Move upon Switch2 closure)

/1s0m50gH02P1000G10R<CR>

/1s0	Store following program in motor number 1 stored string 0 (string 0 is executed on power up).
m50	Set move current to 50% of max
g	Start a loop
H02	Wait for Zero on Switch2 (This is the switch in the starterkit)
P1000	Advance 1000 Encoder Ticks
G10	Repeat loop above 10 times.
R	Run

Type /1e0R or cycle power to execute, since program zero runs on power up.
To Terminate the above loop while in progress type **/1T<CR>**
To clear the above loop from the EEPROM type /1T <CR> followed by /1s0R <CR>

Example #5 (Home to a physical flag)

/1Z100000R<CR>

/1Z100000	Move towards Home while looking for the flag to cut with a max of 100000 steps before declaring an error.
R	Run

Using the opto flag provided in the starter kit issue this command to the drive. Then cut the opto with a steel rule (plastic and paper transmit IR and may not work). The motor will stop and declare the position to be zero. Note that if the opto is cut before issuing this command then the motor will spin in the reveres direction for a max of 10000 steps and then go back into the ipto as soon as it detects that the opto has been cleared. Se also appendix on Homing.

Example #5 (Nested loop example)

/1gA10A1000gA10A100G10G100R<CR>

/1	Talk to motor number 1.
g	Start outer loop
A10	Goto Absolute position 10
A1000	Goto Absolute position 1000.
g	Start inner loop.
A10	Goto Absolute position 10.
A1000	Goto Absolute position 100.
G10	Do inner loop 10 times. (End of Inner Loop)
G100	Do outer loop 100 times. (End of outer loop)
R	Run.

To Terminate the above loop while in progress type **/1T**

Example #6 (Skip / Branch instruction)

/1s0gA0A100S13e1G0R<CR>
/1s1gA0A10S03e0G0R<CR>

Two “Programs” are stored in string0 and string1 and the code switches from one Program to the other depending on the state of input3. In the example given the code will cycle the motor between position A0 and A100 if input3 is High and between A0 and A10 if input 3 is Low. Cycle power or issue /1e0R to run the program.

Stored string 0:

/1	Talk to motor 1
s0	Store following in string0 (executed on power up).
g	Start loop
A0	Goto Absolute position 0
A100	Goto Absolute position 100.
S13	Skip next instruction if 1 (hi) on input 3
e1	Jump to string1
G0	End of loop (infinite loop).
R	Run.

Stored string 1:

/1	Talk to motor 1
s0	Store following in string0 (executed on power up).
g	Start loop
A0	Goto Absolute position 0
A10	Goto Absolute position 100.
S03	Skip next instruction if 0 (low) on input 3
e0	Jump to string0
G0	End of loop (infinite loop).
R	Run.

Type /1T to terminate any loop in progress. Then /1s0R to clear program zero in the EEPROM of desired.

Example #7 (Monitor 4 Switches and execute 4 different Programs depending on which switch input is pushed)

```
/1s0gS11e1S12e2S13e3S14e4G0R<CR>
/1s1A100e0R<CR>
/1s2A200e0R<CR>
/1s3A300e0R<CR>
/1s4A400e0R<CR>
```

Five program strings are stored. Upon power up String 0 automatically executes and loops around sampling the switches one by one, and skipping around the subsequent instruction if it is not depressed. Then for example when Switch1 is depressed stored String 1 is executed, which moves the Servo to position 100. Execution then returns to Stored String 0, due to the e0 command at the end of the other stored strings. If the switch is still depressed it will jump back to String 1 again, but since it is already at that position there will be no visible motion.

To Terminate the above endless loop type **/1T**
Note that using an “e” command to go to another program is a more of a “GOTO” rather than a “GOSUB” since execution does not automatically return to the original departure point.

Example #8 (Move 100 Steps forwards on every rising edge of Switch2)

```
/1gH02H12P100G0R
```

The endless loop first waits for a 0 level on switch1 then waits for a “1” level on Input2. Then A relative move of 100 Steps is issued, and the program returns to the beginning to look for another rising edge.

To Terminate the above endless loop type **/1T**

Example #10 (Hardware locked movement of two motors)

/1gH02P100J3M100J0R –- command to drive with address#1
/2J3gH02P100J3M100J0R –- command to drive with address#2

Using two separate drives the output of one drive is connected to the Switch two input of the other drive. (the negative pin is switched to ground by the J command - see wiring diagram).
The motor one moves then toggles the output then motor 2 moves etc. Note that the initial J3 on motor 2 breaks the H02 on the first drive.

Coordinated motion between multiple axes

For the simple case of motors 1-9, the EZ Servos are addressed as /1, /2, etc. as shown above.
Up to 16 motors can be addressed individually or in banks of 2, 4 ,or “All”, increasing versatility and ease of programming. Synchronized motion is possible by issuing commands addressed to individual EZ Servos without the “R” (Run) command, which sets up the command without executing it. At the proper time, the “R” command is sent to a bank of motors to start several actions in concert.

Addressing motors 10-16
Use the ASCII characters that are the ones above 1-9, which are
10 = “:” (colon)
11 = “;” (semi colon)
12 = “<” (less than)
13 = “=” (equals)
14 = “>” (greater than)
15 = “?” (question mark)
16 = “@” (at sign) – use setting zero on the address switch for this.

For example /=A1000R moves Servo #13 to position 1000.

Addressing banks of motors:
Global addressing of more than one motor is also possible.
The same command can be issued to a bank, or different commands issued to the motors individually, (minus a “Run” at the end) and then a global “Run” command issued to a bank or to All.
The Banks of two are:
Motors 1 and 2 : “A”
Motors 3 and 4 : “C”
Motors 5 and 6 : “E”
Motors 7 and 8 : “G”
Motors 9 and 10 : “I”
Motors 11 and 12 : “K”
Motors 13 and 14 : “M”
Motors 15 and 16: “O”
The Banks of four are:
Motors 1,2,3, and 4 : “Q”
Motors 5,6,7, and 8 : “U”
Motors 9,10,11, and 12: “Y”
Motors 13,14,15 and 16 : “]”- (close bracket)

For All motors:
Use the Global address “_” (underscore).

Example #9 Coordinated Motion with axes doing same motion.

/_A10000R<CR>

/_	(Slash then Underscore) Talk to all 15 Motors.
A1000	Goto Absolute position 1000.
R	Run. All motors will go to Absolute position 1000

Example #10 Coordinated Motion with axes doing different motions

/1A10000<CR>
/2A200<CR>
/AR<CR>

/1A10000<CR>	Set up motor 1 command buffer to go to **A**bsolute position **10000.**
/2A200<CR>	Set up motor 2 command buffer to go to **A**bsolute position **200.**
/AR	Execute current commands in buffer for **Bank Address "A"** which is motors 1 and 2. (The "A" here is an Address of a Bank of motors 1&2 because it comes after the slash and should not be confused with the "A" that means absolute position.) Both moves will start at the same time, and complete at a time determined by the Velocity set for each axis.

APPENDIX 1
SERVO MOTOR ELECTRICAL SPECIFICATION

The EZ Servo® will work with most Servo motors, however the performance achieved will be a function of the motor used.

Select the following:
Kv: Back EMF constant. Chose a motor such that the back EMF at the max speed required is about half of the supply voltage. This will allow for good controllability at the top speed. Motors usually have a voltage specification that is equal to the back EMF at the max speed. So eg use a 12V rated Motor with a 24V supply. (The EZServo regulates the current so the motor will not be damaged.)

L: Motor Inductance, a general guideline is:
For EZSV17 the inductance must be >500uH
For the EZ23 the Inductance must be > 100uH.

Operation of lower inductance motors is possible but depends on exact values of supply voltage, motor voltage, inductance and resistance. Please contact the factory for assistance. Typically motors with inductance less than stated above can be run with the addition of an extra series inductance of about 500uH or so. Inductor must be capable of handling the full motor current.

The EZServo drive is a “Chopper Drive” which works by applying the full supply voltage to the motor until the current reaches the target current, and then switching off the voltage and letting the current decay for a fixed off time of around 3uS. When the voltage is applied the rate of rise of current is dI/dt=V/L. The dI/dt must be less than about 0.25A/uS in the absolute worst case, lowest inductance for the drive to be able to control the current. The application of the full voltage to the drive does not “burn” the motor because it is the IsquaredR heating which “burns” the motor, and since the current is controlled to the value programmed by the “m” command, the motor operates safely.

R: Motor Resistance. In most cases when Kv and L is selected as stated above, this value will be acceptable.

APPENDIX 2
HOMING ALGORITHM IN DETAIL

The "Z" command is used to initialize the motor to a known position. When issued the Motor will turn toward 0 until the home opto sensor is interrupted. If already interrupted it will back out of the opto and come back in until re-interrupted. Current motor position is set to zero. The Homing is done at a speed set by "V".

The maximum number of steps allowed to go towards home is defined by the Z command operand + 400. The maximum number of steps away from home (while sensor is cut) is 10000.

Opto and flag should be set up to be unambiguous, i.e. when motor is all the way at one end of travel, flag should cut the opto, when at other end of travel flag should not cut opto. There should only be one black to white transition possible in the whole range of travel.

To set up the home flags:

1) First ensure that a positive move e.g. /1P100R moves away from home and the home flag. If motor does not move away from home, rephasing of the motor power and hall sensors wires is required to change the direction of rotation. Switch any 2 motor phase wires and then retry all 6 hall sensor combinations, also swap the A and B lines on the encoder to maintain negative feedback..
2) The Default condition expects the output of the Home flag to be low when away from home (as is the case in an opto). If Home flag is high when away from home (as in the case of the "Normally Open" switch) then issue the command /1f1R to reverse the polarity that is expected of the home flag.
3) Homing should be done at a slow speed, especially if homing to a narrow Index pulse on an encoder, which may be missed at high speeds.
4) Issue /1N1R if home is Opto 1.   Issue /1N2R if home is the index on the encoder. Subsequent Z commands will then home to the chosen signal.
5) Note that the "ap" polarity command affects the home sensor polarity also, in a manner similar to the f1 command.

Continued……..

Appendix2 Continued….

Home to Index:
The drive will home to the index with N2 mode. /1N2V10000Z100000R etc. The index pulse is narrow and homing needs to be done slowly since this mode samples the home flag in software. For higher speed hardware based capture see N7 mode below. This mode only works on a normally low signal that goes high at the index.

High Speed Index capture mode:
The drive will home to the index with N2 mode. However some index pulses can be very narrow and may be missed by the sampling type search done by the N2 mode. Starting with firmware V7.02 a new N7 high speed index capture mode has been implemented. In this mode the drive will do a high speed seek for a high going index pulse, and will stop after finding it, and set the index point to zero. The drive will normally stop at a negative value after this type of homing, /1N7Z1000000R etc, but can be brought back to home with the addition of an A0 command after the home. **/1N7Z10000000A0R** etc. This mode only works on a normally low signal that goes high at the index. Note that this mode requires a hardware modification to the board. Please contact factory.

Ensure that motor power wires are kept away or are shielded from encoder and home sensor wires. Noise pickup can cause false home signals, especially in the case of the High Speed Capture. Only ground the home sensor to the sensor I/O connector, which is a quiet ground.

APPENDIX 3
HEAT DISSIPATION

Most applications require intermittent moving of the motor. In the EZ Servo, the current is increased while a move is performed, and the current is then reduced at the end of the move. The dissipation in the drive is proportional to the current flowing in the drive, and therefore the dissipation occurs primarily during the “move”. The one exception to this is the rare instance where the drive has to fight a constant load such as gravity.

When the Drive generates heat, the heat first warms the circuit board and heatfin (if any). Only then does the heat transfer to the surroundings. For intermittent moves that are less than one minute in duration, the Drive primarily cools using this thermal inertia of the board and heatfin, and not by steady state dissipation to the surrounding ambient.

The EZServos are designed to work beyond the voltage and current that they are rated for, however the small size of the boards limit their ability to dissipate heat in steady state. It is recommended that the drive be derated linearly from 100% Duty Operation at 50% current to 25% Operation at 100% of rated current. (Ie if an application requires 100% current (m=100) during the entire move, moves should only performed about 25% of the time – average over about 5 minutes). Typically servos only require significant current when accelerating or decelerating, so even if “m” is set to 100, it does not mean that 100% of current is flowing into the motor during the entire move.

Most applications will not require derating of the drive.

The EZServo will work up to 85C PCB copper temperature, however for long life a PCB copper temperature of less than 55C should be maintained in steady state. Addition of forced air cooling etc can improve the duty cycle of the drive.

The drive should not be mounted inside a sealed enclosure with stagnant airflow. If it is necessary to do this, then the drive should be bolted against an aluminum enclosure with a thermally conductive foam such as BER225 from Digikey / Bergquist filling the space between the drive and the enclosure, ensuring conduction of heat to the enclosure (rather than convection).

APPENDIX 4
OEM PROTOCOL WITH CHECKSUM

The Protocol described in the majority of this manual is DT (Data Terminal) protocol. There is however a more robust protocol known as OEM protocol that includes checksums. AllMotion Drives work transparently in both protocols. And switch between the protocols depending on the start transmission character seen.

The OEM protocol uses 02 hex (Ctrl B) as the start character and 03 Hex (Ctrl C) as the stop character. The 02 Hex Start Character is equivalent to the / character in DT protocol.

OEM PROTOL EXAMPLE1:
/1A12345R(Enter) in DT Protocol is equivalent to
(CtrlB)11A12345R(Ctrl C)# in OEM protocol

Explanation	**Typed**	**Hex**
Start Character	Ctrl B	02
Address	1	31
Sequence	1	31
Command	A	41
Operand	1	31
Operand	2	32
Operand	3	33
Operand	4	34
Operand	5	35
Run	R	52
End Character	Ctrl C	03
Check Sum	#	23

The Check Sum is the binary 8 bit XOR of every character typed from the start character to the end character, including the start and end character. (The Sequence character should be kept at 1 when experimenting for the first time.) Note that there is no need to issue a Carriage return in OEM protocol.

Note that some earlier models require the first command issued after power up to be a DT protocol command. Subsequent commands can be in DT protocol or in OEM protocol. Very early models do not have OEM Protocol implemented at all.

OEM PROTOL EXAMPLE 2:
/1gA1000M500A0M500G10R(Return) in DT Protocol is equivalent to
(CtrlB)11gA1000M500A0M500G10R(CtrlC)C in OEM protocol

The C at the end is Hex 43 which is the checksum (Binary XOR of all preceding Bytes).

Sequence Character:
The Sequence Character comes into effect if a response to a command is not received from the Drive. In this instance the same command can be resent with bit 3 (repeat bit) of the sequence byte set, and bits 0-2 representing the sequence number.

When the repeat bit is set consecutive commands received by the drive must have a different sequence number in order to get executed. (Only the sequence number is looked at – not the command itself)

This covers both possibilities that (a) the Drive didn't receive the command and (b) The Drive received the command but the response was not received.

The sequence number can take the following values.
31-37 without the repeat bit set or 39-3F with the repeat bit set.
(The upper nibble of the sequence byte is always 3.)

APPENDIX 5 BLDC MOTOR WIRING

The procedure below describes how to figure out the phasing of the encoder and the hall sensors. However, please note that AllMotion can perform this process for no extra charge. Just ship us a motor and we will ship it back fully wired with a board.

BLDC Motor Wiring:

First it is necessary to set the phasing of the Hall Sensors:
The phasing of the hall sensors is unrelated to the encoder connection, and no encoder connection is necessary at this time.

1) The procedure is to wire the BLDC motor nominally, as shown in the wiring diagram, and then try all 6 combinations of the 3 hall sensor wires until one combination works.
2) Set the current limit low using /1m10R. (Use of current limited supply set to about quarter of the motor rated current is also recommended).
3) With the encoder unplugged issue the command /1A10000R and the motor should spin smoothly in one direction for about 5 seconds before giving up.
4) While the motor spins hold the shaft of the motor (Only if no danger of injury) and ensure that there are no "Dead Spots" in the torque. Four of the six combinations of the 3 hall sensor wires will have obvious dead spots. Of the remaining two, one will have subtle dead spots and the other will be perfect.
   The Combinations are: 123,132, 213,231, 312,321 .
   Only the Hall sensor wires need to be switched – don't switch power wires.
   Turn Off Power when Switching Wires or Disconnecting Motor. (because the inductance of the motor will create a high voltage spark when the motor is disconnected, which will damage the driver chip)

Second it is necessary to make sure we have negative feedback from the encoder.
First plug the encoder in and rotate the shaft by hand and issue /1?8 . This command retrieves the encoder count, and if the encoder is functioning this count will go up and down as the shaft is turned.
Plug in the encoder and issue a command /1A1000R . If the motor spins endlessly and then gives up, then try again with the encoder A, B channel wires switched.

Motor Tuning:

Typically (in 99% of cases) the motor will be stable with the default constants that are loaded on power up.
If the behavior is oscillatory, then increase the value if the Differential constant.
Eg . /1y3000R
If increasing the Differential term does not work and the motor is noisy and the encoder ticks can be "heard", then simultaneous reduction of both the Proportional and Differential constant is recommended. /1w250y500R works well especially with high (4000) line count encoders.

Motor Overload:
If motor gives up half way through a move, and gives an overload error (Error 9 in Ezcommander or Upper or lower case I in Hyperterminal when queried with /1Q after error) this is due to the fact that the motor could not keep up with the commanded trajectory. Typically increase the value of the move current "m" to allow the motor to move faster, or increase the supply voltage to the motor, or reduce the commanded speed V.

The change in "m" changes the torque and allows the motor to combat friction losses, and also keep up with the commanded acceleration.

Changing the supply voltage when using a encoder feedback controlled motor, should have no effect on the speed of the motor, if it does, then there is not enough voltage to overcome the back EMF of the motor, reduce the commanded speed V or increase the supply voltage.

Continued…………

APPENDIX 5 Continued ……

Motors with no encoders:

In N=0 mode Motors with no encoders will work in velocity control mode in by using the hall sensor crossings as a gage of velocity. In this mode only the Integrator is active and The "x" value controls the acceleration between speeds. (L command is inactive in this mode). The velocity is set in Revs per second using the V command.

In N= 1 mode, It is also however possible to run motors with no encoders by using the Hall sensors as a position encoder, by wiring over two of the hall lines over to the encoder A and B inputs in addition to the regular Hall connection. Any 2 of the hall lines can be wired over, however as with any encoder the A and B lines must be connected to count up when the motor moves in the positive direction.
This mode typically has better performance than the N0 mode in velocity control applications, and can also act as a crude position control mode.
Typically in this mode very low Accelerations and Velocity values must be used since the halls act as a very low line count encoder. Try /1L1V10000R.

When Hall sensors are used for feedback, the PID gain values should be increased from the default values to compensate for the low count rate.

It is also necessary to run the hall sensors on 5V since the encoder connections are 5V. In the EZSV17 which puts out 15V hall power, it will be necessary to use the encoder power at 5V to power the hall sensors.

Analog Hall Sensors:

EZServo is not designed to work with analog hall sensors, however it is possible to put the analog hall sensors into comparators to recreate regular hall sensor waveforms. This is not recommended but is possible.

Brush Motors with no encoders (Requires V7.08D+ firmware):

Brush motors with no encoders can also be operated in velocity mode with some inherent speed regulation. Note that the back EMF "E" of the motor is proportional to the Velocity , and the relationship between applied voltage V and the speed (proportional to E) is given by V= IR + E . Where I is the current and R is the armature resistance. If the motor has a small resistance R of the windings, then V is approximately equal to E and velocity. And a small drop in E can bring up the current I and hence the torque significantly, essentially regulating the speed.
The "ah" command can set the average input V by PWM. The range of ah is 0-1000.
To enable this mode issue for example /1N0m50ah100R . Where N0 sets the mode and m sets the max motor current 0-100%. This command only works in the EZSV23.

APPENDIX 6 BLDC MOTOR TUNING

Typically (in 99% of cases) the motor will be stable with the default constants that are loaded on power up.
If the motor Vibrates or oscillates on power up, try issuing /1w250y500R
The motor should be stable but the response will be somewhat slow.
Increasing both P (w) and D (y) values in proportion will "stiffen" the servo.

If zero following error is needed then the Integrator I (x) will need to be turned on: (eg /1x10R) , however the system will become more unstable and harder to compensate when the integrator is turned on. Try to achieve the best possible result with the P and D values only and then turn on just a little I.
Eg /1w700x10y30000R

The default values work well with 512 –2048 line count encoders. For higher line count encoders use smaller values of PID.

The use of a current limited lab supply is recommended while tuning the motor. Large currents may be drawn if oscillations occur.

Motor Overload:
If motor gives up half way through a move, and gives an overload error (Upper or lower case I when queried with /1Q after error, or "Error 9") this is due to the fact that the motor could not keep up with the commanded trajectory.
This is due to one of two reasons:

1) In the case of a fist time set up, this could be due to incorrect hall sensor phasing.
2) In the case of a fist time set up, this could also be due to the encoder being wired with the A and B lines swapped, thus giving positive instead of negative feedback.
3) Not enough current to overcome steady state losses. Typically increase the value of the move current "m" to allow the motor to move faster.
4) Not enough voltage to overcome the back-EMF of the motor. Once the back EMF equals the applied voltage the drive will limit at this speed. Increase the voltage applied to the drive. For example /1V10000P0R should run at V=10000 regardless of the applied voltage, if increasing the voltage increases the speed, then the motor speed is limited by the applied voltage and the speed is not regulated. Typically the drive will shut down when it detects this condition. Increase the voltage or decrease the speed.

High Speed Trajectory Generator Mode:

Normally the trajectory generator computation is done at 2KHz. However during motion this can cause audible 2KHz tones in high bandwidth systems such as piezo drives. A high speed trajectory generator mode which computes the trajectory at 20KHz can be activated in Software V7.01A and later. This reduces the audible noise. This mode is enabled by the “an32768” mode. The trajectory computations are done 10X faster, and the Velocities will be 10X and the accelerations will be 100X normal mode. Note that this does not change the feedback compensation, as this was and is always done at 20KHz.

Examples in this mode:

/1w100x0y100A0an32768L2V500000gA20000A0G3R

APPENDIX 7

DEVICE RESPONSE PACKET

EZ Servos and EZ Servos respond to commands by sending messages addressed to the "Master Device". The Master Device (which for example is a PC) is assumed always has Address zero. The master device should look for responses starting with /0.

After the /0 the next is the "Status Character" which is actually a collection of 8 bits.
These Bits are:
Bit7 … Reserved
Bit6 … Always Set
Bit5 … Ready Bit - Set When EZ Servo is ready to accept a command.
Bit4 … Reserved

Bits 3 thru 0 form an error code from 0-15
0 = No Error
1 = InitError
2 = Bad Command (illegal command was sent)
3 = Bad Operand (Out of range operand value)
4 = N/A
5 = Communications Error (Internal communications error)
6 = N/A
7 = Not Initialized (Controller was not initialized before attempting a move)
8 = N/A
9 = Overload Error (Physical system could not keep up with commanded position)
10 = N/A
11 = Move Not Allowed
12 = N/A
13 = N/A
14 = N/A
15 = Command Overflow (unit was already executing a command when another command was received)

Continued…

Example Response to command /1?4

FFh:	RS485 line turn around character. It's transmitted at the beginning of a message.
2Fh:	ASCII "/"Start character. The DT protocol uses the '/' for this.
30h:	ASCII "0" This is the address of the recipient for the message. In this case ASCII zero (30h) represents the master controller.
60h:	This is the status character (as explained above
31h:	
31h:	These two bytes are the actual answer in ASCII. This is an eleven which represents the status of the four inputs. The inputs form a four bit value. The weighting of the bits is: Bit 0 = Switch 1 Bit 1 = Switch 2 Bit 2 = Opto 1 Bit 3 = Opto 2
03h:	This is the ETX or end of text character. It is at the end of the answer string.
0Dh:	This is the carriage return…
0Ah:	...and line feed.

A program that receives these responses must continuously parse for /0 and take the response from the bytes that follow /0. The first Character that comes back may be corrupted due to line turn around transients, and should not be used as a "timing mark".

APPENDIX 8
AUTO RECOVERY OF A STALLED SERVO

The EZServo determines a stalled or overload condition, by checking to see if is following a commanded trajectory. If the following error is too great for a period given by the "u" command then the servo will shut down and report an error condition.

Typically the following error is great due to the fact that either:

1)"m" value (current) set too low.
2)"L" value (acceleration) set too high, for Torque available from motor.
3)"V" value (Velocity) set too high, for Torque available from motor.
4) Physical obstruction, or excessive friction.

When an overload condition is detected it will be reported back as an upper or lower case I when the status is quizzed. This status can be used by an external computer to execute a recovery script.
However it may be desired that the servo recover by itself in the case of a stand-alone application. For this purpose we have the "n" mode bits n512, and n1024 .
Depending on which of these bits is set the servo will execute stored program 13, 14 or 15 when an overload is detected. Program12 is also executed as a last resort if programs 13,14,15 cannot auto recover by a after retrying a number of times given by the "au" command.

Example:
Set m=5 so that we can stall a motor easily.
Set u=1000 so the motor overloads after 1 second.
Set au = 10 so that 10 retries max are allowed
Set n =512 so that Stored Program 13 will be issued on error condition.
/1s0m5n512au10u1000e1R
/1s1gA10000A0G0R
/1s13r1e1R
/1s12z333R
Program zero enables Auto-recovery mode and calls program 1 on power up.
Program 1 cycles between 10000 and 0
When a stall occurs while running program 1, program13 is automatically called. This program recovers the servo to the current encoder position and then calls program 1 again. The servo will attempt auto-recovery 10 times as given by the "au" command, then, if no move completes after 100 retries, it will execute stored program 12.
In this example cycle the power and the servo will go between 0 and 10000. Now hold the shaft of the motor and stall it (if safe to do so). The motor can be felt to retry the move 10 times prior to setting its position to 333 and stopping.

Note that /1n8R mode can also be used instead of /1r1R. In this mode the encoder position is never reset and a stalled servo is automatically turned on to the current position when a new A, P or D command is received.

APPENDIX 9
ANALOG INPUTS AND ANALOG FEEDBACK
Available on Software version 3.90 and later

ANALOG INPUTS
The 4 Inputs on the EZServo are all ADC Inputs.

1) These ADC values can be read via RS485. E.g**. /1?aa<CR>.** These values are on a scale of 0-16368 as the input varies from 0-3.3V. The inputs as shipped are good to about 7 bits, but can be made to be better than 10 bits with the removal of the input over-voltage protection circuitry, (call factory for details).
2) The threshold upon which a Digital "One" or "Zero" is called can be varied with the "at" command and hence affect the Halt H command or Skip S command. **E.g. /1at309999R<CR>.** – This sets the threshold on input 3 to be 09999. Note that it is necessary to insert leading zeros after the Input Number (3) since the threshold value must always be entered as 5 digits (00000-16368).
3) The thresholds for all 4 inputs can be read back with the **/1?at<CR>** command. The units have a default threshold value of 6144 = (1.24V).
4) It is possible for example to regulate pressure, by turning a pump on or off depending on an analog value read back , by setting the threshold of the One/Zero call to be the regulation Point. E.g. /1at308000gS03P1000G0R
5) A potentiometer can be placed as shown in the wiring diagram and its position read back via the /1?aa<CR> command. Note that the supply provided (which normally drives an LED) has 200 ohm in series to 5V, so the use of a 500 ohm potentiometer will give almost a 0-3.3V range on the inputs.

POTENTIOMETER POSITION COMMAND
Potentiometer 2 can be used to command the position of a motor. The value read back on potentiometer 2 from 0-16368 is multiplied by the multiplier "am" and then divided by 256, then an offset given by "ao" is added. The motor will then use this number just as if it had been commanded by a /1A12345R type command. There is further a "deadband" command "ad" which sets a dead band on the 0-16368 pot value read back such that a value outside this deadband must be seen before a command is issued to cause a move.

Simple Example:
Issue /1A0n8192R<CR> enables potentiometer command mode. Apply a voltage to the Opto2 input pin between 0 and 3.3V. The /1?0 position will track the /1?aa value.

Detailed Example:
Stepper position = ((analog value from pot / 256) x ("am" multiplier)) + ("ao" offset value)

Use 500 Ohm Potentiometer.
Supply pin of pot already has 200 Ohm in series on the board to 5V
Value from pot = 0 to 16368 for 0-3.3V on wiper.
Issue /1A0R to turn on the servo
Issue /1ao1228R<CR> sets ao offset to 1228 . Default is 0
Issue /1am256R<CR> sets am multiplier to 256. Default is 256.
Issue /1ad100<CR> sets dead band on pot to 100 . Default is 50
Then issue /1n8192R to enter the mode.

Note that the servo must be on and the velocity and acceleration set to desirable values for the response desired./1A0L10000V1000000R etc

POTENTIOMETER POSITION FEEDBACK

Potentiometer 1 can be used as an encoder. The value read back is from 0-16368.
/1N3R enables Potentiometer 1 as the encoder, (instead of the quadrature encoder).

To make this mode work:

1) Use a 500 Ohm Pot wired in to the Potentiometer 1 position (see wiring diagram).
2) Connect motor shaft to shaft of pot.
3) Issue /1w200x0y500R<CR> to reduce PID values.
4) Issue /1N3r1R<CR> to enable the pot as encoder and turn the servo on.
5) The shaft of the motor should now be holding position.
6) If motor slams to the end of the pot, then reverse the pot ends. (positive feedback).
7) Issue a /1?aa<CR> which will return the pot value. Issue a /1?<CR> which will return the motor position. The pot value and the motor position should now be the same.
8) Issue /1A2000R<CR> then issue /1A4000R<CR> The motor should move between 1/8th and ¼ of the pot full range (16368).
9) It may be necessary to have stops such that the pot ends do not take the full force of the motor, if the motor is commanded to the ends.

POTENTIOMETER VELOCITY COMMAND (Joystick Mode)(Firmware V 6.91+)

Potentiometer 2 can be used to command velocity of the drive.
Velocity in encoder ticks/sec =(Pot 2 Value – (deadband/2)) x (gain-multiplier / 256)

To enter this mode typically hook up a 500 Ohm Pot to Opto2 as shown in the wiring diagram . Then set it to mid position and issue **/1ad100am1000n65536z0P0R**

“ad” = dead-band and “am” = gain-multiplier must be set prior to entering the mode. If needed issue /1n0R then /1T to disable the mode and then change these values as necessary.

A zero z value in the above string will set will set the zero velocity position to be the current pot position of the pot. Any non zero z value will cause that value from the pot to be considered the zero velocity point.

To exit this mode typically type /1n0R to remove the mode and then type /1T to terminate any move in progress.

Note that it is possible to issue point to point moves when in potentiometer velocity mode. /1ad100am1000n65536z0gA1000A0GR will move between position 1000 and 0 with the magnitude of the velocity (but not the direction) defined by the potentiometer. In this mode it may be better to use z1 to define the end of the potentiometer as zero velocity, rather than use z0 which defines the current potentiometer value to be the zero value.

Starting with V7.X+ additional features have been added :
/1an2R (software limits) or /1n2R (hardware limits) turns on limits in potentiometer velocity mode, The underlying P0 command will be terminated if a limit is hit. If automatic restart upon hitting a limit is desired then issue /1an16R in addition which automatically reissues the P0 command when a limit is hit, thus allowing the joystick to drive the device into the limits repeatedly without halting. In this mode the limits will need to be moved to the switch inputs since the potentiometer is wired to what is normally a limit input. /1an16384R moves the limits to the switch inputs. an16 + an16384 = an16400.

Example #1
Set 1.5V into Input#4
From Powerup Issue:
/1A0an16402aaP400aaD-400n65636z0P0R
A0 turns on servo
an16402 = 16384 + 16 + 2 = move limits to switches + auto restart + software limits
aaP aaD sets up limits to be +/-400
n65536 turn on potentiometer velocity mode
z0 sets current coltage to be zero velocity
It can be observed if voltage on input is moved away from the zero setpoint then the motor moves and stops at the limits +/-400 + decel distance

Example#2
/1A0an16400f1n2n65536z0P0R

APPENDIX 10
SPECIAL CONSIDERATIONS WHEN DRIVING THE NANOMOTION ST1, HR1, HR2 OR HR4 MOTORS

Driving the Nanomotion HR1, HR2 or HR4 Drives require V7.0 or greater version of software.
The Nanomotion Drive is operated in closed loop mode with an encoder feedback only. For best performance an encoder of 10um to 0.1um resolution (Eg Renishaw RGH24Z15D00A) or similar is recommended. The encoder should give quadrature pulses of 0-5V levels. The encoder can be run non differential for short cable runs (ie use only A+, B+ and I+), but may require differential line receivers for long encoder cable runs.

Servo Tuning:
The Nanomotion Piezo servo can be controlled with the standard PID algorithm together with another algorithm designed to mitigate the deadband in the motor.

The PID values are set by the commands w,x and y. Typically set the integrator to be zero, as it will interfere with the deadband mitigation algorithm.

First adjust the PID values to get the final position error small, and yet have the system stable. Typically for a HR2 motor running a 1um encoder, set **/1w200x0y200R**
Increase w and y to reduce the position error. Then issue the deadband mitigation commands, ai and af, ai sets the strength of the deadband mitigation algorithm, and af sets the how often the algorithm runs. Higher values of ai increase the strength, and af sets the period between the algorith running with af=1 being 400mS, af=2 being 800mS. Typically set **/1ai160af10R**. A value of af=0 turns off the deadband mitigation algorithm.

High Speed Trajectory Generator Mode:
Normally the trajectory generator computation is done at 2KHz. However during motion this can cause audible 2KHz tones in high bandwidth systems such as piezo drives. A high speed trajectory generator mode which computes the trajectory at 20KHz can be activated in Software V7.01A and later. This reduces the audible noise. This mode is enabled by the "an32768" mode. The trajectory computations are done 10X faster, and the Velocities will be 10X and the accelerations will be 100X normal mode. Note that this does not change the feedback compensation, as this was and is always done at 20KHz.
Examples in this mode:
/1w100x0y100A0an32768L2V500000gA20000A0G3R

Continued….

THE NANOMOTION HR1, HR2 OR HR4 MOTORS Continued….

Overload timeout and max current setting:

The Nanomotion HR1, HR2 or HR4 Motors Drives cannot be driven at full current indefinitely. It is necessary to limit the max current allowed to a safe value and also amount of time the max current is applied for to a few seconds only. Else the motors can be damaged.

The max current should be reduced from the allowed (default) 100% m=100 to a value that is enough to get the job done, plus allow a margin of safety. So start with m= 25 for example and work upwards. (m=100 is default). Also set the overload timeout "u" value to be a few seconds down from the default 30000 milliseconds (u=30000 default). So for example try **/1m25u1000R** when beginning to experiment. This will set 25% current max for one second before the servo is shut off.

High Speed Index capture mode:

The drive will home to the index with N2 mode. However some index pulses can be very narrow and may be missed by the sampling type search done by the N2 mode. Starting with V 7.02 a new N7 high speed index capture mode has been implemented. In this mode the drive will do a high speed seek for a high going index pulse, and will stop after finding it, and set the index point to zero. The drive will normally stop at a negative value after this type of homing /1N7Z1000000R etc, but can be brought back to home with the addition of an A0 command after the home. **/1N7Z10000000A0R** etc. Note that this mode requires a hardware modification to the board. Please contact factory.

APPENDIX 11 ADDRESSING MORE THAN 16 MOTORS ON THE SAME BUS

		ADDRESS BANK SELECTION
		Requires Version V7.07 or later
aB	aB49520 aB49521 aB49522	Use this command <u>ONLY</u> if you have more than 16 motors on the bus. Else <u>DO NOT</u> use it. Allows upto 48 drives to be addressed by adding 16 or 32 to the value of the address switch. Sets address bank to 0-15 (normal default mode) Sets address bank to 16-32 (uses ascii 61-70 for addressing) Sets address bank to 33-47 (uses ascii 71-7f for addressing) If the bank is unknown issue /_aB49520 to globally set all addresses to bank zero.
		BINARY ADDRESS SELECTION
		Requires Version V7.07 or later
aB	aB496XXX	Use this command <u>ONLY</u> if you have more than 16 motors on the bus. Else <u>DO NOT</u> use it. It is possible to set the address of a device to any binary number between 0 and 127 decimal. XXX will be the number in decimal. For example /1aB496065R will set the address of the device to 65 which is “A” in ascii. After sending this command further communication is by using “A” as the address. The device now responds to /A&. Typically this command would be stored in the EEPROM eg /1s0aB496065R If address is lost issue /_aB49520R to globally set all addresses to bank zero. The underscore “_” global character decimal 095 is not allowed with the aB496XXX command

APPENDIX 11
Setting Limits of motion

Hardware Limits:
Enabled by /1n2R
The two opto sensors should be placed at the ends of travel. Opto1 at the low end of travel and Opto2 at the high end of travel. These can be replaced with any device that provides TTL like signal levels. The Threshold as to what is considered a high or low can be set with the /1at command.
/1f0n2R Turns on the limits with the limits being low when not active, and high when the limits are reached.
/1f1n2R turns on limits with the limits being high when not active, and low when the limits are reached.
To disable limits type /1n0R.

Using Switches as limits
/1an16348 moves the hardware limits from the two optos to the two switches

Software Limits (V7.08+):
Enabled by /1an2R
It is also possible to set up Software Limits where an internal check is done in software to against a constant which the motion controller will not exceed.
/1aaD1000 sets the lower limit to +1000
/1aaP4000 sets the upper limit to +4000
example /1aaL65000aaD-10000aaP10000an2gA200000A-200000GR
will only go between 10000 and –10000 even though the loop commands 200000
To disable limits type /1an0R.

Deceleration once limits are reached are set by the aaL command /1aaL100R etc.
The L command will over write the aaL command and the aaL must be set after the L command eg /1L1000aaL10000R .

Note that the device "homes" to the lower limit. It is necessary to turn off limit mode when homing, because the homing algorithm goes past the limit and then backs up to the edge.

RELEASE NOTES V7.08H

NOTE: WE WILL SHIP V7.08H AS STOCK FROM 10/1/2013 ONWARDS PLEASE REQUEST V5.45 IF YOU HAVE RELEASED YOUR APPLICATION WITH THIS PRIOR FIRMWARE.

Every effort has been made to maintain backwards compatibility between V5.45 and V7.08H, however we do not guarantee it.

CHANGES FROM V5.45 TO V7.08H

1) Negative numbers: commands such as A-1000 V-1000 P-1000 can be issued.
2) On the fly position and velocity change. The target position and velocity can be changed while a motion command is in progress. The commands must be issued one at a time an “immediate” command.
3) Potentiometer velocity mode. Similar to a joystick where velocity is proportional to voltage.
4) Separate accel L and decel aL values. Also separate aaL value for decel when a limit is hit.
5) Addition of PWM to Outputs see aK and aaK command
6) Software limits. See aaP and aaD command in appendix11
7) Addition of new modes to “n”, “an” and “N” commands