SMS Series of Linear Actuators Stepper Command Set

Command Set for Stepper Models: EZHR17EN and EZHR23ENHC

Document Revision: 8/26/08

SMS Series of Linear Actuators User's Manual February, 2011

Command Syntax

Commands to the EZ Stepper 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 to be in one command. You can also create loops in the strings and cause the EZ Stepper to become a stand-alone device that responds to switch inputs.

Finally, storing such strings into the onboard EEPROM allows the EZStepper 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 EZStepper can even be commanded from a serial enabled PDA.

COMMAND SET *(Also see examples on page 10)*

Positioning Command

Positioning Command	Command*	Operand
Move Motor to absolute position *(microsteps or quadrature encoder ticks-32 Bit Positioning).*	A	0-(2^31)
Move Motor relative in positive direction *(microsteps or quadrature encoder ticks)* 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** command or by a falling edge on the Switch 2 Input.	P	0-(2^31)
Move Motor relative in negative direction *(microsteps or quadrature encoder ticks)* *(Note: for a finite move, the ending Absolute position must be greater than zero).* 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** command or by a falling edge on the Switch2 Input. **Eg. /1D10000R** – note ending position must be > 0 – Note that negative positions are allowed in SoftwareVersion 6.7 and above eg /1A-1000R	D	0-(2^31)

Homing Command

Homing Command	Command*	Operand
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. **Eg /1Z300000R** *See Appendix 2 for further details.*	Z	**0-(2^31)** (400)
Change current position without moving Sets current position to be position specified without moving motor. New microstep position *(preferably)* should have the same remainder as old position, when divided by 1024, else the motor may physically move/lose up to 2 steps when this command is issued.**Eg. /1z65536R**	z	0-(2^31)

*Commands are case sensitive

Home Flag polarity	**f**	**0 or 1** (0)
Sets polarity of home sensor, default value is 0. *(See Appendix 2)* **Eg /1f1R**		
Change direction of rotation considered positive	**F**	**0 or 1** (0)
This should only be done once on power up. Do not use if in Encoder Feedback mode. **Eg /1F1R**		

Set Velocity Commands

	Command*	Operand
In Position Mode Set Max/Slew Speed of motor	**V**	**1-2^24** (305064)
Sets microsteps per second. It is recommended that this drive be left in 256 micro-step mode. *(Since very high microsteps/sec numbers can be issued)* **Eg /1V100000R** If the encoder ratio (aE command) is set, then the units of velocity change to be Encoder counts / second.		

Set Acceleration Commands

	Command*	Operand
In EZHR17EN Set Acceleration factor	**L**	**0-65000** (1000)
Accel in microsteps / sec^2) = (L Value) x (400,000,000/65536). **Eg. using t=V/a /1L1R takes 16.384 Seconds to get to a speed of V=100000 microsteps/second** (**Note:** *The acceleration does not scale with encoder ratio.)* B Bump jog distance see "n1" command		
Bump	**B**	
jog distance see "n1" command		

Looping and Branching Commands

	Command*	Operand
Beginning of a repeat loop marker	**g**	
See examples below on how to set up a loop. **Eg. /1gP10000M1000G10R**		
End of a repeat loop marker	**G**	**0-30000**
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. /1gP10000M1000G10R**		
Halt	**H**	
current command string and wait until condition specified.		
Wait for low on input 1 *(Switch 1)*		01
Wait for high on input 1 *(Switch 1)*		11
Wait for low on input 2 *(Switch 2)*		02
Wait for high on input 2 *(Switch 2)*		12
Wait for low on input 3 *(Opto 1)*		03
Wait for high on input 3 *(Opto 1)*		13
Wait for low on input 4 *(Opto 2)*		04
Wait for high on input 4 *(Opto 2)*		14

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 /1gH02P10000G20R** - waits for switch 2H command with no number after it waits for Switch 2 Closure (low).

*Commands are case sensitive

Skip next instruction depending on status of switch	**S**	
Skip next instruction if low on input 1 *(Switch 1)*		01
Skip next instruction if high on input 1 *(Switch 1)*		11
Skip next instruction if low on input 2 *(Switch 2)*		02
Skip next instruction if high on input 2 *(Switch 2)*		12
Skip next instruction if low on input 3 *(Opto 1)*		03
Skip next instruction if high on input 3 *(Opto 1)*		13
Skip next instruction if low on input 4 *(Opto 2)*		04
Skip next instruction if high on input 4 *(Opto 2)*		14

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. **Eg /1gS02A10000A0G20R** - skips on switch 2.
Also see "H" command for I/O dependant program execution.

Program Storing and Recall

	Command*	Operand
Stores a Program	**s**	0-15
0-3 or 0-15 depending on model, Program 0 is executed on power up. *(25 full commands max per string)* **Eg. /1s1A10000A0R** Note: This command takes approx 1 Second to write to the EEPROM.		
Executes Stored Program	**e**	0-15
0-15. **Eg. /1e1R**		

Program Execution

	Command*	Operand
Run	**R**	
the command string that is currently in the execution buffer. **Eg. /1R**		
Repeat Run	**X**	
the current command string		

Set Max Move Current / Hold Current

	Command*	Operand
For Steppers " Move" Current on a scale of 0 to 100% of max current	**m**	**0-100** (25)
100% = 2A for EZHR17EN. **Eg. /1m40R**		
Sets "Hold" Current on a scale of 0 to 50% of max current	**h**	**0-50** (10)
100% = 2A for EZHR17EN **Eg. /1h15R**		
Reserved	**aw**	
Reserved	**ax**	
Reserved	**ay**	

Miscellaneous

	Command*	Operand
Ping Command (lower case "p")	**p**	**0-65000**
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. **Note:** *Care must be taken when using this command because it can tie up the 485 bus.*		

*Commands are case sensitive

SET MICROSTEP RESOLUTION / ENCODER

SET MICROSTEP RESOLUTION / ENCODER	Command*	Operand
Adjusts the resolution in micro-steps per step. It is recommended that step resolution be left at 256 microsteps *(default)*. *(It is recommended that this drive can be left in 256 micro-step mode, only use reduced resolution if step and direction mode (n96) is selected and high frequency step pulses cannot be generated.)* For best micro step results, a motor must be selected that is capable of micro step operation.	j	**2, 4, 8, 16, 32, 64, 128, 256** (256)
Special Modes	**N**	**1-5** (1)
1 = Encoder With No Index *(Default)*. Homes to Opto.		
2 = Encoder With Index. Homes to Index.		
3 = Uses Potentiometer 1 as an encoder. See Appendix 9.		
4 = First Home to Opto Then home to Index *(future Release)*		
5 = CANBUS Slave mode. In this mode the unit is a slave to position messages received via the CANBUS EZLink Connector. *(future release)*		
Sets Modes – Interpret as combination of Binary Bits	**n**	**0-4095** (0)
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.		
Bit1: /1n2R Enable Limits. *(The two optos become limits switches)*. The polarity of the limits is set by the "f" command		
Bit2: /1n4R Enable Continuous Jog Mode. Continuous run of motor while switch is depressed. Velocity is given by "V" command.		
Bit3: /1n8R Enables Position Correction Mode. See Appendix 8		
Bit4: /1n16R Enabled Overload Report Mode. See Appendix 8		
Bit5: /1n32R Enable Step And Direction Mode if (1) or enable Dual Encoder Mode if (0) Eg /1n96R<CR> (96=32+64) Enables step and dir mode and slaves the motor to it. *(/1?10 Reads the count)* See Appendix 14		
Bit6 : / 1n64R Enable Motor slave to encoder/step-dir.		
Bit7 : /1n128R . Used for Joystick Mode, See Appendix9		
Bit8 : /1n256R When Set this bit will disable the response from the drive. *(Future Release)*		
Bit9 and Bit10: When set these bit will execute one of the stored programs 13, 14 or 15 whenever the feedback shuts down the drive due to an overload or an error. ("au" retries are exhausted. See Appendix 8 /1n512R will execute program 13 /1n1024R will execute program 14 /1n1536R will execute program 15 See Appendix 8 for an example.		
Bit11 /1n2048R Reserved. Bit12 /1n4096R Reserved.		
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. See also "d" command.		
Bit15: /1n32768R When Set this bit will kill any move if switch 2 is pushed. See also "d" command.		

*Commands are case sensitive

Bit16: **/1n65536R** When Set Potentiometer on Opto 2 input will set velocity. *(Joystick Mode)* See Appendix 9.		
/1an16384 switches the limits from the main axis from the two opto inputs *(inputs 3,4)* to the two switch inputs (inputs 1,2). Example /1an16384R. The second axis limits remain unchanged at 1,2. *(Software V 6.998+)*	**an**	**16384**

POSITION CORRECTION / FEEDBACK MODE

See Appendix 8 for position correction commands.

Miscellaneous	Command*	Operand
Adjustable baud rate Eg /1b19200R This command will usually be stored as program zero and execute on power up. Default baud rate is 9600. *(note: correct termination and strict daisy chaining required for reliable operation at higher baud rates)*	**b**	**9600** **19200** **38400** **to 230400** (9600)
Allows the user to correct any unevenness in micro-step size. It is best to adjust this with a current probe, but adjusting for lowest audible noise is a good approximation. This command can be executed while the motor is running. Try values very near 1500 like 1470.	**o**	**0-3000** (1500)
Wait M number of Milliseconds.	M	**0-29999**
Set Jog distance when in n1 Mode	**B**	**0-2^31**
Processor Reset *(Available in V6.7+)* This command will initiate a Processor Reset the same as that which happens on power up. 5073 is chosen to avoid inadvertent resets. Eg: /1ar5073R<CR>	**ar**	**5073**
Response delay *(Available in V6.79+)* This command allows the delay from the controller receiving the command to the response being sent out to be programmed. Eg: /aP1000R<CR> sets the delay to 1000mS. (measured in Milliseconds)	**aP**	**0-30000** (5)
Switch Debounce value Applies to kill move command only, The switch one or two must be depressed for a period of this number x 50uS prior to a Kill Move being called.	**d**	**0-65000** (10)
Backlash compensation When a non zero value of K is specified, The drive will always approach the final position from a direction going more negative. If going more positive the drive will overshoot by an amount K and then go back. By always approaching from the same direction the positioning will be more repeatable.	**K**	**0-65000** (0)

*Commands are case sensitive

SINUSOIDAL SCAN *(Available in V5.1 +)*		
Sets amplitude of scan. See Appendix 10	**aA**	**0-2^31**
Sets frequency of scan. See Appendix 10	**aW**	**0-2^31**

ON/OFF POWER DRIVER		
On/Off Driver – Interpret as 2 bit Binary Value, 3=11= Both Drivers On, 2=10=Driver2 on Driver 1 Off etc.	**J**	**0-3** (0)

STEPPER DRIVE DAUGHTER CARD COMMANDS		
/1aM1R Selects first drive. From then on all commands are sent to drive 1. /1aM2R selects drive2. **See Appendix 11**	**aM**	1 - 2
The Second drive homes to opto #2 or Switch#1		

BIDIRECTIONAL DRIVE DAUGHTER CARD		
/1l80R sets current to 80% (Lower case L)	**l**	0-100
/1O1R sets the current flow one way	**O**	1
/1I1R sets the current flow in the other direction	**I**	1

DEVICE RESPONSE PACKET		
See Appendix 7 for detailed description of device response to commands		
Some commands are new and present only in later models.		
Equipment Solutions reserves the right to enhance the specifications at any time		

ANALOG TO DIGITAL CONVERTER COMMANDS

All 4 Inputs are ADC inputs and can be read and acted upon by the program. *Please see Appendix 9*

	Command*	Operand
Reads back all 4 Input ADC Values E.g. /1?aa<CR> The Readback order is channels 4:3:2:1	**?aa**	
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.	**at**	**100000 to 116368** **200000 to 216368** **300000 to 316368**
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. This can be used for example to regulate pressure to a given level, by turning a motor on/off at a given voltage.		**400000 to 416368** (6144)
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.*		
Reads back the thresholds for all 4 channels. The Readback order is channels 4:3:2:1 **Eg /1?at<CR>**	**?at**	

*Commands are case sensitive

POTENTIOMETER POSITION COMMAND

The motion of the Stepper can be slaved to value read from Potentiometer2. *Please see Appendix 9.*

After multiplication by the am value this offset is added to obtain the position command. Eg **/1ao1000R<CR>**	**ao**	**0-20000** (0)
The Pot value is multiplied by this value and divided by 256 to get the position command. Eg /1am512R<CR>	**am**	**0-20000** (256)
Sets a deadband (in microsteps) around the pot value used for the last move, which must be exceeded before a new move command is issued. The deadband is measured in micro-steps and will need to be increased as the gain is increased. **Eg /1ad100R<CR>**	**ad**	**0-20000**(50)

Hardware protocol: The EZ Stepper communicates over the RS485 bus at 9600 baud, 1 Stop bit, No Parity, No flow control.

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

IMMEDIATE QUERY COMMANDS

These commands do not require an "R" at the end

Terminate current command or loop. eg /1T ?	**T**	
0 Returns the current Commanded motor position eg. /1	**?**	**0**
Reserved	**?**	**1**
Returns the current Slew/Max Speed for Position mode	**?**	**2**
Reserved	**?**	**3**
Returns the status of all four inputs, 0-15 representing a 4 bit binary pattern. Bit 0 = Switch1 Bit 1 = Switch2 Bit 2 = Opto 1 Bit 3 = Opto 2	**?**	**4**
Returns the current Velocity mode Speed *(Stepper Only)*	**?**	**5**
Returns the current step size microsteps /step *(HR Version Only)*.	**?**	**6**
Returns the current 'o' value. *(HR Version Only)*	**?**	**7**
Returns Encoder Position. *(can be zeroed by "z" command)*	**?**	**8**
Erases all stored commands in EEPROM.	**?**	**9**
Returns the second encoder (n=0) / or step and dir input (n=32) count.	**?**	**10**
Note that it is possible to just count pulses on switch inputs with this mode. (Future Release)		
Returns the current Firmware revision and date	**&**	
Query current status of EZ Stepper/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 EZStepper/EZServo is ready to accept commands. It is cleared when the EZStepper/EZServo is busy. The least significant four bits of the Status byte contain the completion code.	**Q**	

*Commands are case sensitive

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. See Appendix 7		
The n mode works in both immediate mode and in strings.	**n**	
/1$<CR> Returns the currently executing command string.	**$**	

**Micro steps for HR version is in whatever resolution currently selected using "j" command.

EXAMPLES OF COMMAND STRINGS IN DT PROTOCOL:

Example #1 (A Move to Absolute Position)

/1A12345R<CR>

This breaks down to:

1. "/" is the start character. It lets the EZ Steppers 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 Absolute position 12345
4. "R" Tells the EZ Stepper to Run the command that it has received.

<CR> is a carriage return that tells the EZ Stepper that the command string
is complete and should be parsed.

***Note:** Hyperterminal issues each character as you type it in. Therefore it is not possible to cut and paste in Hyperterminal. Backspace is allowed only upto the address character. If backspace is used, all characters "backspaced" must be retyped in. If a typing error is made, typically hit enter and type it all in again – what was typed in will be overwritten as long as the R command at the end was not present.*

Example #2 (Move loop with waits)

/1gA10000M500A0M500G10R<CR>

This breaks down to:

1. "/" is the start character. It lets the EZ Steppers 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 Absolute position 10000
5. "M500" causes the EZ Stepper to wait for 500 Milliseconds.
6. "A0" makes the motor turn to Absolute position 0.
7. "M500" is another wait command for 500 Milliseconds.
8. "G10" will make the string repeat 10 times starting from the location of the small "g"
9. "R" Tells the EZ Stepper to Run the command that it has received.
10. <CR> is a carriage return that tells the EZ Stepper that the command string
is complete and should be parsed.

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

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).

To erase a program, store the program without any commands eg. **/1s0R**

Example #4 (Set Current, Wait For Switch 2 closure, Home to Opto)

/1s0m75h10gJ3M500J0M500G10H02A1000A0Z10000R<CR>

Store following program in motor number 1 stored string 0 *(string 0 is executed on power up).*	**/1s0**
Set move current to 75% of max	**m75**
Set hold current to 10% of max	**h10**
Start a loop	**g**
Turn on both on off drivers.	**J3**
Wait 500 mS J0 Turn off both on off drivers.	**M500**
Wait 500 mS G10 Repeat loop above 10 times.	**M500**
Wait for a switch2 input to go low.	**H02**
Move to position 1000	**A1000**
Move to position 0	**A0**
Home the stepper to opto #1 Max Steps allowed to find opto = 10000.	**Z10000**
Run	**R**

Example #5 (Nested loop example)

/1gA1000A10000gA1000A10000G10G100R<CR>

Talk to drive number 1.	**/1**
Start outer loop	**g**
Goto Absolute position 1000	**A1000**
Goto Absolute position 10000.	**A10000**
Start inner loop.	**g**
Goto Absolute position 1000.	**A1000**
Goto Absolute position 10000.	**A10000**
Do inner loop 10 times. (End of Inner Loop)	**G10**
Do outer loop 100 times. (End of outer loop)	**G100**
Run.	**R**

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

Example #6 (Skip / Branch instruction)

```
/1s0gA0A10000S13e1G0R<CR>
/1s1gA0A1000S03e0G0R<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 codewill cycle the motor between position A0 and A10000 if input3 is High and between A0 and A1000 if input 3 is Low.

Stored string 0:

Talk to motor 1	**/1**
Store following in string0 (executed on power up).	**s0**
Start loop	**g**
Goto Absolute position 0	**A0**
Goto Absolute position 10000.	**A10000**
Skip next instruction if 1 (hi) on input 3	**S13**
Jump to string1	**e1**
End of loop (infinite loop).	**G0**
Run.	**R**

Stored string 1:

Talk to motor 1	**/1**
Store following in string0 *(executed on power up)*.	**s0**
Start loop	**g**
Goto Absolute position 0	**A0**
Goto Absolute position 1000.	**A1000**
Skip next instruction if 0 (low) on input 3 e0 Jump to string0 G0 End of loop *(infinite loop)*.	**S03**
Run.	**R**

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

```
/1s0gS11e1S12e2S13e3S14e4G0R<CR>
/1s1A1000e0R<CR>
/1s2A2000e0R<CR>
/1s3A3000e0R<CR>
/1s4A4000e0R<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 Switch 1 is depressed stored String 1 is executed, which moves the stepper to position 1000. 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 1000 Steps forwards on every rising edge of Switch 2)

```
/1gH02H12P1000G0R
```

The endless loop first waits for a 0 level on switch1 then waits for a “1” level on Input2. Then A relative move of 1000 Steps is issued, and the program returns to the beginning to look for another rising edge.

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

COORDINATED MOTION BETWEEN MULTIPLE AXES

For the simple case of **motors 1-9**, the EZ Steppers 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 Steppers 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 stepper #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 square bracket)*

For All motors:

Use the Global address "_" *(underscore).*

For addressing more than 16 motors:

It is possible to have add an off set of 16 or 32 to the number on the address switch.
Please see the "aB" command for how this works.

Example #9 Coordinated Motion with axes doing same motion	
/_A10000R<CR>	
(Slash then Underscore) Talk to all 15 Motors.	**/_**
Goto Absolute position 1000.	**A1000**
Run. All motors will go to Absolute position 1000	**R**

Example #10 Coordinated Motion with axes doing different motions	
/1A10000<CR> **/2A200<CR>** **/AR<CR>**	
Set up motor 1 command buffer to go to Absolute position 10000.	**/1A10000<CR>**
Set up motor 2 command buffer to go to Absolute position 2000.	**/2A2000<CR>**
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.	**/AR**

APPENDIX 1: STEPPER MOTOR ELECTRICAL SPECIFICATION

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

A Stepper Motor moves by generating a rotating magnetic field, which is followed by a rotor. This magnetic field is produced by placing a Sine Wave and a Cosine Wave on two coils that are spaced 90 degrees apart. The torque is proportional to the magnetic field and thus to the current in the windings.

As the Motor spins faster, the current in the windings need to be changed faster in a sinusoidal fashion. However the inductance of the motor will begin to limit the ability to change the current. This is the main limitation in how fast a given motor can spin.

Each winding of the motor can be modeled as an Inductor in series with a resistor. If a step in Voltage is applied then the current will rise with time constant L/R. If L is in Henrys and R is in ohms then L/R is the time it takes in Seconds for the current to reach 63% of its final value.

Note: *There is also the Back EMF of the motor, which essentially subtracts from the applied voltage.*

The current I for a step function of voltage V into a coil is given by

$$I = (V/R)\ (1-\exp(tR/L))$$

This equation is a standard response of a first order system to a step input. The final value of current is seen to be V/R. (This system is similar to a spring (L) in parallel with a Damper (R) being acted upon by a Step in Force (V) giving a resulting velocity (I).)

There are two methods by which the current can be made to change faster.

(1) Reduce the Inductance of the motor

(2) Increase the forcing function voltage V.

For (1) it is seen that for high performance, a motor with low inductance is desired.

For (2) the trick is to use a motor which is rated at about ¼ of the supply voltage (V). This means that it takes less time to ramp the current to a given value. *(Once the current reaches the desired value the “Chopper” type drive used in the EZ Steppers ® will “Chop” the input voltage in order to maintain the current. – So the current never actually gets to the final value of V/R but the advantage of “heading towards” a higher current with the same time constant means that the current gets to any given value faster.)* The lower voltage motor also has less back EMF, and does not subtract as much from the applied voltage.

So for example, for a 24V supply, use a motor rated at around 6V, and then use the “m”, and “h” commands to set the current regulation at or below the rating for the motor. The default values on power up are h=10% and m=25% and should be safe for most motors.

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 the current speed “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.

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 go away from home, flip the connections to only one of the windings of the stepper.

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) Issue the command eg. /1Z100000R or /1f1Z100000R as necessary.

4) 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. 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. Home can be done to an opto (N1 Mode) or Index Pulse (N2 Mode).

The Main axis homes to opto1 (input #3).This opto is also the lower limit.The Main axis uses opto 2 (input #4) as its upper limit.

The Second axis on the daughter card uses the two switch inputs as home and limits.

Switch 1 (input1) is Home and lower limit.

Switch 2 (input2) is the upper limit.

Note that limits are engaged by /1n2R. *(lower case “n”).* The default (n0) mode does not check the limits when moving.

Default “f” mode (f0) expects the inputs to be low when the axis is away from the limits/home. /1f1R reverses this. “f1” can be chosen per axis eg /1aM1f1aM2f0R selects different polarities for the home flags of the two different axes.

Further the threshold for the inputs Hi/Low transition level can be programmed via the “at” command. Thus, if for example the home sensor does not fully pull to a TTL low level, as in the case of say a reflective sensor. The intermediate level can be accommodated by the EZStepper circuits, without external signal conditioning.

Also the optos and switches are interchangeable. If four optos are desired , power for the extra optos can be drawn from the 5V supply on the encoder connector. These extra optos may require an external resistor in series with the LED. When connecting switches connect between any input and ground.

Main Axis Homing Details:

There are four full steps in a single electrical cycle that moves the stepper motor. (A+, B+, A-, B-). For repeatability in homing, the home position is set to first step in that cycle that occurs after the flag edge has been seen. *(This means that the home position is defined some ways beyond the middle of the flag).*

However there is a small but finite chance that an ambiguity in home position may occur in the rare case that the exact point of switching into A+ occurs at the same point at which the flag gets cut. In which case a 4 step ambiguity in home position may exist, because sometimes the flag may cut just before and sometimes just after. The procedure below describes a method by which the ambiguity can be removed. However, this procedure need not be followed if a 4 step inaccuracy in Home position is acceptable.

To eliminate the home position ambiguity. First issue the Z command, allow the motor to home. Then move 2 full steps *(in any direction),* now mechanically move the flag edge *(or sensor)* such that it trips in the middle of the sensor by adjusting it while watching the status LED on the board which shows the status of the home sensor. This will ensure that the flag trips at A- and thus the motor will home to a unique position of A+.

Another way to do this, if it is not hazardous, is to put the motor in an endless homing loop "**/1gZ10000GR**", then move the flag/opto around while the motor is homing. It will be noticed that the motor will home to two distinct positions that are 4 steps apart. Make sure the Hi to Low transition point of the Opto is NOT near these positions *(exact position does not matter as long as it is not near the place where it homes to).*

Second Axis Homing Details:

The second axis will home to the exact transition of the home flag, and does not seek a Phase A zero. The second axis uses the switch inputs for homing and limits.

Manual homing:

Motors can be manually homed to any input by the use of a polling loop such as:

/1s1z0R - store set current position to zero program 1 **/1z100000gD1S04e1GR** - go backwards in an endless loop until input 4 goes high.

Homing to a hard stop:

It is possible to send the motor against a hard stop and then force the position counter to zero or some other value as necessary. **/1z0R** zeros the position counter and encoder for the motor that is currently selected (lower case "z"). **/1z333R** will set the position to 333 etc. So **/1aM1z0R** zeros the counters for motor 1 and /1aM2z0R zeros the position counters for motor 2.

APPENDIX 3: MICROSTEPPING PRIMER

First lets consider a Full Stepping Driver:
A Stepper motor moves by having two windings that are orthogonal to each other and sequencing the current in these windings.

When full stepping a typical sequence is:

A+	(Only winding A current applied in +ve Direction)
B+	(Only winding B current applied in +ve Direction)
A-	(Only winding A current applied in -ve Direction)
B-	(Only winding B current applied in -ve Direction)

A full electrical cycle consists of 4 steps.

It can be seen that if the windings are not physically placed orthogonally then the 4 steps may not be of equal size, and the delta in motion will only be a constant if the number of steps is divisible by four, even when in full step mode.

Now consider Microstepping:
Microstepping is achieved by placing two sinusoidally varying currents that are 90 degrees apart, in the windings of the stepper. This causes a torque vector of equal length to rotate causing smooth inter step motion of the rotor.

However in order to get even motion in every step it is necessary

1) That the windings be mechanically orthogonal
2) That the windings produce equal torques for equal currents.
3) That there is no other “detent torque” that is acting upon the rotor in the absence of current. This detent torque is easily felt by rotating the stepper *(with windings disconnected and not shorted).* A motor that is good for microstepping will feel smooth when rotated by hand *(somewhat like a DC motor)* with little tendency to detent.
4) The current not be so small that the driver cannot regulate it to the microstepping accuracy desired. In general most inexpensive stepper motors cannot microstep with any accuracy. Typically, a special motor designed for microstepping must be run at a significant current in order to get even microsteps Typically the move current must be set equal to the hold current, when accuracy is required, because if the current is reduced at the end of the move, the motor will fall back into a detent position.

In general most inexpensive stepper motors cannot microstep with any accuracy. Typically, a special motor designed for microstepping must be run at a significant current in order to get even microsteps. Typically the move current must be set equal to the hold current, when accuracy is required, because if the current is reduced at the end of the move, the motor will fall back into a detent position.

APPENDIX 4: HEAT DISSIPATION

Most stepper applications require intermittent moving of the motor. In the EZ Stepper, the current is increased to the "move" current, the move is performed, and the current is then reduced to the "hold" current (automatically). The dissipation in the drive is proportional to the current flowing in the drive, and therefore the dissipation occurs primarily during the "move".

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 electronics for the EZ Steppers are fully capable of running at the rated voltage and current. However due to the small size of the boards, which limits the steady state heat transfer to the ambient, care must be taken when the drive is used in high duty cycle and/or high current applications. For conservative operation it is recommended that the duty cycle be reduced linearly, from 100% duty at 50% of rated current, to 25% duty at 100% of rated current. *(Duty cycle means the percentage of the time that the drive is moving the load – average over 5 minutes).* Conservatively, the maximum continuous run at 100% current is about 1minute. An on board thermal cutout typically trips after about 2 minutes at 100% current. *(This cutout is s elf-resetting when the drive cools).* Of course, at 50% of current, the drive will run continuously with no time limit.

Most "intermittent on" applications will NOT require derating of the drive.

In case the EZ Stepper is required to run 100% of the time at 100% current, forced air cooling or bolting against an aluminum hetafin with a heat conductive foam is recommended.

EZSteppers are designed with parts rated at 85C or better. This means the PCB copper temperature must remain below 85C.The ambient air temperature allowed depends on the airflow conditions.

MTBF is 20000Hrs at 85C PCB copper temperature and doubles for every 10C lower than 85C.

APPENDIX 5: STEP LOSS DETECTION USING OPTO

For some applications, which operate without encoder feedback, it may be necessary to detect loss of steps due to the mechanism stalling for any reason.

Step loss is easily detected by following the algorithm below:

1) Home the Stepper to the Opto using the **Z** Command.

2) Move out of the flag a little by issuing say a **A100** Command

3) Figure out the exact step on which the Flag gets cut by issuing **D1** commands followed by ***/*1?4** commands to read back the Opto. Let's call this value Y.
*(This only needs to be done once during initial set up)***.*

4) Execute the move sequence for which step loss detection is needed.

5) Issue a command to go back to absolute position Y+1

6) Check the opto, it should not be cut *(read Opto back with /***1?4** *command).*

7) Now issue a command to go to position Y-1.

8) Check the opto, it should be cut *(read Opto back with /***1?4** *command).*

9) If the Opto was not at the state expected , then steps may have been lost.

10) Step loss detection can also be done by looking for changes on the other inputs.

**** See Homing algorithm detail in Appendix2.*

APPENDIX 6: 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. Equipment Solutions 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 EXAMPLE 1:

/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)1**1gA1000M500A0M500G10R(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 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 parse the communications on the bus continuously for responses starting with /0. *(Do NOT for example look for the next character coming back after issuing a command because glitches on the bus when the bus reverses direction can sometimes be interpreted as characters)*

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)*

Note that in the RS485 Bus devices must respond right away, after the master sends a command, before the success or failure of the execution of the command is know. Due to this reason some error messages that come back are for the previous command, example "failure to find home".

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".

Example Initialization Error Response:
Note that the Upper Nibble only typically takes on values of 4 or 6 (Hex) An initialization error has response has 1 in the lower Nibble. So the response is 41 Hex or 61 Hex which corresponds to ASCII character upper case "**A**" or lower case "**a**", depending on if the device is busy or not.

Example Invalid Command Response:
Note that the Upper Nibble only typically takes on values of 4 or 6 (Hex) An invalid command has response has 2 in the lower Nibble. So the response is 42 Hex or 62 Hex which corresponds to ASCII character upper case "**B**" or lower case "**b**", depending on if the device is busy or not.

Example Operand Out of range Response:
*Note that the Upper Nibble only typically takes on values of 4 or 6 (Hex) An operand out of range has response has 3 in the lower Nibble. So the response is 43 Hex or 63 Hex which corresponds to ASCII character upper case "***C***" or lower case "***c***", depending on if the device is busy or not.*

Example Overload Error Response:

Note that the Upper Nibble only typically takes on values of 4 or 6 (Hex) An overload error has response has 7 in the lower Nibble. So the response is 47 Hex or 67 Hex which corresponds to ASCII character upper case "I" or lower case "i", depending on if the device is busy or not.

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.

APPENDIX 8: POSITION CORRECTION MODE & OVERLOAD REPORT MODE

Position Correction Mode (Requires V6.7 or later Software):
Position correction mode, when enabled will move until the encoder reads the correct number.
Once enabled Positions are given in Quadrature Encoder Counts of the encoder (Not in microsteps).
If the motor stalls during a move then this mode will reattempt the move until the encoder reads the correct number. These algorithms run while the stepper is in motion and will detect a stalled motor during a move. There are two main types of feedback arrangements:

The first method is to place the encoder on the motor shaft.

The second method is to place the encoder on the component that is finally positioned, and may be only be loosely coupled to the motor due to backlash etc.

The EZStepper will work with either type of feedback.

Procedure for setting up encoder feedback:

First Set the Encoder ratio:

Encoder Ratio = *(MicroSteps/Rev Divided by Quadrature Encoder Ticks/rev) x 1000*
This must preferably be a whole number after the multiply by 1000. (If this is not a whole number, then see discussion further down).

The Motor must be left in 256th Microstep mode for correct operation of feedback mode. For Example A 1.8 Degree Stepper *(running in 1/256th step mode),* which has 200x256 Microsteps /Rev is Hooked up to a 400 line encoder which has 1600 uadrature encoder counts.

aE= ((200x256)/(400x4))x1000=32000 Issue the command **/1aE32000R** to set this Encoder ratio.

If the encoder ratio is unknown do the following:

Leave the Encoder ratio at its power up default of 1000.

Ensure that the encoder increases its count when the motor moves in the positive direction.
If not, switch either the AB lines on the encoder which will reverse the count direction
OR switch the wires to one of the windings on the stepper motor, which will reverse the direction of rotation. (Do not use the “F” command). Issue **/1n0R** to clear any special modes.

Issue **/1z0R** which will zero both the encoder and step position.
Issue **/1A100000R** and ensure that the move completes at a velocity that does not stall.
Issue **/1?0** which reads back the current position – This should be **100000**.
Issue **a /1?8** which reads back the encoder position.
Issue **a /1aE0R** which auto divides these two numbers.
Issue **/1?aE** which reads back the encoder ratio computed.

This value read back is only a rough guide and will be out by a few counts due to inaccuracies in the motor position and run-out in the encoder.The value read back MUST be overwritten by the EXACT value that represents ratio.

Now Issue **/1aE32000R** or whatever exact number represents the encoder ratio.

Second: *(optional)* **set the Error in Quadrature Encoder Ticks allowed before a Correction is issued:**

Eg /1aC50R *(default is 50)*

Third: *(optional)* set the Overload Timeout Value:

This is the number of retrys allowed under a stall condition.

Eg /1au10000R *(default is 100)*

Fourth: Enable the Feedback mode:

Zero positions just prior to enabling feedback mode the by issuing /1z0R. (Or issue **/1z10000R** etc if you need to at this time) Enable Position Correction Mode by issuing /1n8R.

Example: (V6.7+)
/1z0aC50h40m40au100aE32000V1000n8R will start position correction mode.

Fine position correction (V6.99+):
When in n8 mode there are two position correction algorithms, the COURSE ALGHORITHM as described above, operates to bring the motors to within the value given by "**aC**" (upper case C). This algorithm runs all the time *(while the stepper is in motion)* and will detect a motor that is stalled or lagging by more than "**aC**" during a move. When a problem is detected the axis will stop and reissue a move starting from zero velocity so as to slowly spin up motor that may have stalled at high speed.

In Version 6.99 and greater, a fine position correction integrator algorithm (FINE ALGORITHM) is engaged once the major move is completed to zero out any small residual position error. The Speed of this correction is affected by the "x" integration period value. /1x10R is default. Smaller values of x correct faster but may lead to oscillations eg /1x3R. The fine position correction deadband is set by "**ac**" *(lower case c)* eg: /1ac10R, the default ac is 20 encoder counts. This can be adjusted down to zero, if desired but it may take time to settle depending on stiction and backlash, especially if the encoder is decoupled from the motor. The integration algorithm runs on hold current and this may need to be adjusted to a reasonably high number, depending on the load. It is also best to run hold current equal to move current such that the motor does not relax and detent at the end of the move.

Move complete *(non busy)* will be asserted when the move completes to within "**ac**" for the first time. Subsequent disturbances greater than "**ac**" but less than "**aC**" will be corrected by the fine correction algorithm and will not be reported as "busy". Only disturbances greater than "**aC**" will result in the coarse correction algo being engaged and busy being asserted.

Example: (V6.99+)

/1aM1z0aC50ac10x10h40m40au100aE32000L100n8R

Notes on feedback mode:

If motor consistently stops during a move ….
If a very fine line count encoder is used such that for example the encoder ratio is around 2000, or if the encoder is decoupled from the motor shaft , or if the encoder ratio has some fractional component and is non integer, then increase the error (**aC**) allowed for the coarse algorithm say set **aC** to 2000. This way a move that is in progress will not be halted and restarted because the course algorithm detects hat the following error is too large. Instead the move will complete with some error and the fine algorithm integrator will null out any error to within the lower case "**ac**" value at the end of the move.

Busy/Non Busy Status:
The status will read busy until any move is reached to within the "**ac**" value, however once a move is complete any subsequent disturbance is handled by the fine integrator as long as it is less than "**aC**" *(bigC),* these subsequent corrections by the fine algo will not read busy, however if the error gets to be greater than "**aC**" then the dead reckoning algo is turned on, and a formal correction move is initiated. This move changes the drive status to busy and no external commands will be accepted during this time. It is important to set the coarse dead reckoning deadband *(aC value)* to a reasonable number say 50) else the drive will always be attempting to correct.

Other Notes:

(1) When position correction mode is enabled, **/1n8R**, then the drive will keep retrying any stalled moves, until "**au**" retries are exhausted, and will NOT halt any strings or loops upon detection of a stall.

(2) During position correction mode **/1T** will halt any move, but there is a possibility that the drive may instantly reissue itself a position correction command, especially if it is fighting a constant disturbance. It may be necessary to issue a **/1n0R** to positively halt a move in progress.

(3) If the encoder ratio is changed from its default of 1000, the allowed max position will be decreased from +2^31 by the same ratio. The count will rollover from positive to negative range when this limit is exceeded.

(4) Do not use the upper case "**F**" *(reverse direction)* command when using encoder feedback mode, instead switch the encoder AB lines or the wires to one phase of the motor.

(5) Jog mode will not work with encoder feedback on.

AUTO RECOVERY IN FEEDBACK MODE Available in V6.997+

Auto Recovery Scripts *(Available in V6.997+):*
In n8 mode the EZStepper determines a stalled or overload condition, by checking to see if the encoder is tracking the commanded trajectory. If the encoder is not following the commanded trajectory within the error specified by "ac"then a number of retries given by the "au" command is tried.

When an overload condition is detected (retry counter has exceeded the "**au**" value) it will be reported back as an upper or lower case I (Error 9) 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 drive recover by itself in the case of a stand-alone application. For this purpose we have the "n" mode bits n512, and n1024.It is necessary to combine these bits with n8 so for example n512+n8 = n520. 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. Note overload error on any motor will execute error recovery if enabled.

Example:

/1s13p1202n520R – set error recovery script to send “1202” on every recovery.

/1s12p1201R – set final script to send “1201”

/1aM1m5h5z0aC50au5u3aE12800L100n520R

Set **h= 5** and **m=5** so that we can stall the motor easily.

Setup encoder ratio as appropriate: **/1aE32000R** etc.

Zero Encoder and Position Counter **/1z0R**

Set **au = 5** so that 5 retries max are allowed

Set **u = 5** so that program13 is run a max of 3 times.

Set **n =520 = n8 + n512** so that Stored Program 13 will be issued on error condition.

Now move the motor shaft so that the motor tries to correct 5 times and then gives up. After the 5th retry the motor will execute stored program13 and will attempt send a 1202 on the 485 bus, then the feedback turns on and 5 more tries are made. If the motor is held stalled then the 1202 will be sent 32 times followed by a 1201 as the final recovery script , stored program 12 is run. T
Typically reasons that the following error is too great are:

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.

POSITION CORRECTION MODE COMMANDS

When in position correction mode, set distance allowed to move before the motor corrects using encoder feedback. **Eg /1aC100R** *See Appendix 8*	**aC**	**1-65000** (50)
Set Encoder ratio. This sets the ratio between the encoder ticks/rev and the microsteps/rev for the motor. **Eg /1aE12500R** *See Appendix 8*	**aE**	**1000-10^6** (1000)
Set Overload Timeout. This sets the number of times the move is retried in case a move stalls. **Eg /1au10000R** When the au retries are exhausted the drive will drop out of feedback mode (n8) and report Error 9 (Overload)	**au**	**1-65000** (10)
Set Integration period. In Version 6.99 and greater, a fine position correction integrator algorithm is engaged once the major move is completed to zero out any small residual position error. The Speed of this correction is affected by the “x” integration period value. /1x10R is default. Smaller values of x correct faster but may lead to oscillations eg /1x3R.	**x**	**1-10000** (10)

POSITION CORRECTION MODE COMMANDS (continued)

Bit3 : /1n8R Enables Position Correction Mode. *See Appendix 8* **Bit4** : /1n16R Enabled Overload Report Mode. **Bit9 and Bit10** : When set these bit will execute one of the stored programs 13, 14 or 15 whenever the feedback shuts down the drive due to an overload or an error. ("au" retries are exhausted. *See Appendix 8* /1n512R will execute program 13 /1n1024R will execute program 14 /1n1536R will execute program 15	**n**	**0-4095** (0)
Sets the final error allowed by the fine position correction algorithm.	**ac**	**1-65000** (50)
In Version 6.99 and greater, sets the number of times error recovery scripts 12, 13 or 14 are run prior to final recovery script 12 is called upon.	**u**	**1-65000** (0)

APPENDIX 9: ANALOG INPUTS AND ANALOG FEEDBACK ANALOG INPUTS

ANALOG INPUTS

The 4 Inputs on the EZStepper 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 *(see wiring diagram)* can be used to command the position of the main axis only *(Motor1)*. 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.

/1aM1ao100ad100am1000n8192R<CR> enables potentiometer command mode.

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

Use 500 Ohm Potentiometer. (See wiring diagram)

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.

Eg **/1ao1228R<CR>** sets ao offset to 1228 . Default is 0

Eg **/1am256R<CR>** sets am multiplier to 256. Default is 256.

Eg **/1ad100<CR>** sets dead band in microsteps to 100 . Default is 50

POTENTIOMETER VELOCITY MODE (JOYSTICK MODE) *Available in Software V6.7+*

Potentiometer 2 (see wiring diagram) can be used to command the velocity of the main axis only (Motor1). The value read back on potentiometer 2 from 0-16368 is multiplied by the multiplier "**am**" and then divided by 256. The motor will then use this number just as if it had been commanded by a **V** command. /**1n65536R** to enter velocity mode. Once in this mode **/1z0R** will set zero velocity to the current position on the pot. Then issue **/1P0** to start an endless move based on the velocity as read from the pot. **/1zxxxR** where xxx is non zero will set zero to that value of the pot. ad sets the dead band on the pot about mid scale am sets the multiplier where:velocity in microsteps/sec or encoder ticks/sec = [(pot value(0to16368) – *(pot zero value from "***z***" command) - ("***ad***" deadband value/2)]* x *(am/256)* x *(65536/20000)*

So, **/1ad100am1000n65536z0P0R** starts the mode.

To terminate type **/1n0<CR> /1T<CR>**

The actual Velocity can be read back by **/1?V<CR>**

It is also possible to use the potentiometer to set the magnitude of the velocity and Switch1 input to be the direction of the velocity. Bit 7 Enables this.

/1ad100am1000n65664z0P0R starts the mode. (65664=65536+128)

Please request V6.998 or later software for this feature.

Further it is possible to know if the shaft is following the commanded velocity, buy using a shaft encoder for feedback and setting the encoder ratio (**aE**) and the following error (**aC**), and setting n16 mode to report overloads 65552 = (65536 + 16). /**1ad100aC400am128n65552aE12800z0P0R**

In the case that Potentiometer velocity mode is to be used with limits, the limits can be switched to the two switch inputs, so as not to interfere with the potentiometer. The command to do this is **/1an16384R**.

Example **/1ad100am1000an16384n65538z0P0R** Switches limits to the alternate inputs, and enables limits (65536 +2 =65538)

POTENTIOMETER POSITION FEEDBACK *Available in Software V6.7+*

Potentiometer 1 can be used as an encoder for the main axis only (Motor1). The value read back is from 0-16368. *Please read Appendix 8 on Encoder Feedback Mode first.*
The operation is identical except that the Pot Acts as an encoder which generates positions between 0 and 16386

/1N3R enables Potentiometer 1 as the encoder, *(instead of the quadrature encoder).*

To make this mode work:

1) Use a 500 Ohm Linear Taper Pot wired in to the Potentiometer 1 position.
2) Connect motor shaft to shaft of pot. Such that a move in the positive direction for the motor increases the value read from the pot using the **/1?aa** command.
3) Turn pot all ALL the way to zero by issuing a **D** command eg **/1D1000R.**
4) Issue a **/1z0R** to zero the motor position.
5) Issue **/1N3R** to enable the pot as encoder.
6) Move motor by issuing P commands until it is about ¾ of pot range.
7) Issue a **/1aE0R** to automatically work out the “encoder ratio” for the pot.
8) Issue **/1n8R** to enable feedback mode.
9) Note that if the motor shaft is forced that a correction will be issued to return the motor to its original position.
10) If shaft oscillates at some positions this may be because of a nonlinearity in the pot, try increasing the dead-band set by the **aC** command. Or it could be because the zero position was not set correctly. Another cause of oscillations is a “scratchy” pot, try adding a 0.1uF capacitance between the wiper terminal and ground.
11) If instead it is desired to start feedback at a non zero pot position then read the current pot value (position) with **/1?aa**, say its 2345, then issue **/1z2345N3n8R**

Example:
(V6.7+) /1aM1z0aC50h40m40au100aE32000V1000N3n8R

Example:
(V6.99+)
With fine position correction **/1aM1z0aC50ac10x10h40m40au100aE32000L100N3n8R**

See Appendix 8 for parameter explanation.

To get out of this mode type **/1n0R** followed by **/1T**

APPENDIX 10: SINUSOIDAL SCAN

SINUSOIDAL SCAN *(Requires Software V6.7 or later)*

The EZHRXXEN Models can be commanded to automatically scan in a sinusoidal profile.

The Amplitude of the scan Motion is set by the "**aA**" command.

Eg. To set amplitude **/1aA51200R** sets a peak to peak amplitude of 51200 microsteps (one rev on a 1.8 degree stepper). Any change in amplitude will be made as the sinusoid goes through the zero cross.

The frequency of the scan is set by the "**aW**" command Eg. To set frequency **/1aW1000R** sets frequency to be f= (X)*20000/ *(1024*65536)* where X is 1000 in this example

To Exit from this mode simply issue a zero amplitude **/1aA0R**

APPENDIX 11: DUAL AXIS DAUGHTER CARD

The EZHR17EN Model has the capability to accept one of three daughter cards.

DUAL AXIS STEPPER DAUGHTER *(Requires Software V6.79 or later)*

The Dual Stepper Daughter card is a 1 Amp 1/16th step capable daughter card. **Note:** *this card only works from a 10V-30V supply.* To issue commands to the daughter card simply issue the command /1aM2R (Motor 2), all subsequent commands go to the daughter card motor. **/1aM1R** returns to the primary axis. Eg **/1aM2V10L1A1000A0R**

The second axis can run, Full, Half, Quarter, and 1/16th step only (**/1aM2j16R** etc).
The default resolution is half step.

The second axis daughter card is rated at One Amp.

The Second Axis Homes to Switch1, and uses Switch1 and 2 as its limits (**/1n2R** mode).
It is possible to home to the index on the channel 2 encoder by issuing the command **/1aM2N2R**.
Then the second axis will home to Opto 2 where the second axis encoder index is connected.
Note: *Beta versions of software before V6.74 always homed to Opto2.*

The second axis and the second encoder can be zeroed by issuing /1aM2z0R The **/1?0** and **/1?8** position queries report the position of the current axis and encoder selected by the last /1aM2R or **/1aM1R** command

Simultaneous Motion:
Starting in Software V6.78 and later it is possible
to run both axes simultaneously in full encoder feedback mode It is possible to issue A,P and D commands simultaneously to both axes, (only these 3 commands work with the following examples: Commands are issued separated by a comma.

/1A1000,-1000R Will mode move axis one to 1000 and axis two to – 1000 /1P-1000, **1000R** Will move axis one backwards by 1000 and axis 2 forwards by 1000.

Dual Axis Feedback Mode Example:
Both axes can simultaneously feed back from encoders or even potentiometers.

/1aM1j256au10000aE32000n8R sets up encoder feedback mode for axis 1

/1aM2j16au10000aE4000n8R sets up encoder feedback mode for axis 2

Note that when feed back is used that the encoder inputs use up the limit switches for the second axis. In case the drive needs to be "Homed" when in feedback mode then one of the other inputs will need to be used, and a manual home routine written. Manual homing can be done by sampling one of the inputs. Eg use stored program zero as an escape from an endless loop

/1s1R - store nothing in program 1 /

1gD1S04e1GR - go backwards in an endless loop until input 4 goes high.

BI-DIRECTIONAL DRIVE DAUGHTER *(Requires Software V3.85or later)*

The Bidirectional daughter card can drive a bipolar current, for example to operate a bidirectional latching relay.

/1l80R sets the current in this drive to 80% for example. (lower case **L**)

/1O1R sets the current flow one way.

/1I1R sets the current flow in the other direction.

It is not necessary to issue Dual axis command **s** such as **aM2** when using this daughter card.

LOGIC OUTPUT DAUGHTER CARD

There are 4 logic signals that come into the daughter card via the miniature DSUB on the main board. These have been put through 24mA capable drive chips and made available on a connector by this daughter card. The signals will respond to commands designed for the other daughter cards, and thus can be used to drive an external drive for example.

The signals are:

STEP – step pulses that are approx 1uS wide are present when axis 2 moves.

DIRECTION – Hi/Low level on this pin sets the direction of motion on axis 2.
This can also be modified by the **O/I** command

CURRENT PWM – 20KHz PWM whose duty cycle responds to commands that set current eg “**m**”, “**h**” for axis 2 , and “**l**” for the bi-directional daughter.

STEP RESOLUTION – activated by **j2 /j16** command.

APPENDIX 12: ON THE FLY POSITION CHANGE AND ANY COMMAND ANY TIME

The EZHR17EN Software version 6.9981 and later allows on the fly position, velocity, and acceleration change. This allows virtually any trajectory to be generated.

Once moving commands can be issued ONE AT A TIME without the "**R**"

/1A10000000R<CR> will start the move

/1V10000<CR> will change the velocity while moving

/1A0<CR> will make the drive automatically decelerate and then head back to zero.

Note that issuing a P200 command for example, if issued while moving, will cause the motor to go 200 steps from the current position (not final target).

A, **P**, **D**, **L**, and **V** can be changed on the fly.

APPENDIX 13: ADDRESSING MORE THAN 16 MOTORS ON THE SAME BUS

ADDRESS BANK SELECTION Requires Version V6.7 or later

Use this command **ONLY** if you have more than 16 motors on the bus. Else **DO NOT** use it. Allows up to 48 drives to be addressed by adding 16 or 32 to the value of the address switch.	**aB**
Sets address bank to 0-15 (normal default mode)	**aB49520**
Sets address bank to 16-32 (uses ascii 61-70 for addressing)	**aB49521**
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.	**aB49522**

APPENDIX 14: ENCODERS AND STEP/DIR PULSE INPUT

The EZHR17EN Model had Dual encoder inputs. Any Quadrature encoder wit AB and (optional) Index input is acceptable. In addition step and direction style position counting is also supported on the secondary encoder.

READ ONLY MODE:

In the simplest mode, the encoder are simply be read back **/1?8** reads the primary encoder on the 5 pin connector, and /1?10 reads back the secondary encoder on the 8 pin connector. The Primary encoder is always in quadrature encoder mode and expects quadrature pulses on the AB lines. The secondary encoder can be placed in quadrature encoder or in step and direction counting mode. Depending on if **/1n32R** mode is enabled or not. Additionally, **1aM1R** and **/1aM2R** switches back and forth between the Primary and Secondary Motors, and /1?8 will read back the primary encoder when in aM1 mode and the secondary encoder when in aM2 mode.

ENCODER /STEP AND DIRECTION FOLLOWING MODE:

In this mode the motor takes the count from the secondary encoder and uses this count as a commanded position. The count can come in from another AB quadrature style encoder or from step and direction pulses. Select AB or Step/Dir with **/1n32R** etc, and then turn on “Motor Slave to Encoder2” by **/1n64R** mode. So for example **/1n96R** (96=32+64) enables step and dir mode and slaves the motor to it. The input step count (readable by /1?10) is related to the motor position by the following relationship.

Stepper position = (“am” multiplier / 256) x Input Step Count) Counts are in microsteps
Eg /1am256n96R

MAIN AXIS ENCODER FEEDBACK MODE:

The Main axis can use the primary encoder for feedback and the secondary encoder for “command”. Please see the section on “Encoder feedback”.

DUAL AXIS FEEDBACK MODE

The main axis can use the primary encoder for feedback and the second axis can use the second encoder for feedback. Please see the section on “Dual Axis”.

ELECTRICAL

Please see the wiring diagram for the wiring details.The Encoder(s) must draw a total current of <100mA from the 5V pin. Encoders must have 0.2V Low to 4V High swing at the input of the connector.

APPENDIX 15: JOG MODES AND LIMIT SWITCHES

JOG:

The EZHR17EN can be placed in a mode that will allow the two switch input to “Jog” the motor backwards and forwards. The command for this is **/1n4R**. Once issued the motor can be moved by pressing switch 1 or 2. Internally these inputs are “pulled up” with 10K to 3.3V and a switch closure to ground is all that is required.

*Note that these are inputs 1 and 2 and the status of the switches can be read via software by /***1?4** *which returns a binary number between 0-15 which represents the status of the 4 inputs.*

LIMIT MODES:

The EZHR17EN uses the two opto inputs as limits. These are inputs 3 and 4. It should be noted that these inputs are general purpose inputs and can be driven by switch closures etc or any device that produces a voltage change. The inputs are actually ADC inputs and the One/Zero threshold can be set by the “**at**” command.

Input 3 is the lower limit and also the home switch. Input 4 is the upper limit. Both limits are simultaneously turned on by the **/1n2R** command. The default expects optos which are low when away from the limit, this polarity can be changed by issuing **/1f1R**, such as when normally open switches are used for limits. *(***Note:** *normally closed switches are better for limits, since any disconnect of the wires will shut down the motion)* When n2 mode is engaged the motor will not move in the direction in which a limit is active, but will back out of the limit.

In V 6.998 and later software the limits can be moved to the switch inputs 1,2 if desired, by issuing the command **/1an16384R**.

Note that the limit inputs can be used to “kill moves” and the execution will

NOISE CONSIDERATIONS:

The inputs are relatively high impedance at 10K and will pick up noise if bundled with the motor wires etc. For long cable runs each input line should be shielded. The addition of a 0.1uF ceramic capacitor from the input to ground at the Board connector may be an alternative to shielding but may slow the response.