Stage Controller
Operation and Programming
Manual

Conix Research, Inc.
857 28th st.
Springfield, OR 97477 USA
Phone: (541) 747-8512
Fax: (541) 747-8528
Web: www.ConixResearch.com
E-mail: Support@ConixResearch.com

Last Revision: October 29, 2008
Version HJ 4.0

MOTORIZED STAGE CONTROLLER

CONTENTS

INTRODUCTION

The Conix Motorized Stage Controller is designed to allow the electronic control of a stage and a Z-focus system.

FEATURES OF THE MOTORIZED STAGE CONTROLLER:

- Natural feeling joystick control of speed, with speed proportional to joystick deflection. This permits easy selection of speeds from as slow as single-step "jogging" to full turbo traverse speed
- Three user-selectable joystick speed sensitivities
- housing Cast aluminum controller case shields against RFI radiation from internal microprocessor
- Table-top control unit size is 7"D x 10.5"W x 3"H
- Compact control unit size is 7"D x 11"W x 2"H
- LED display of X, Y, and Z Coordinates
- USB (COM Port) serial communications
- 3 Axis coordinated motion.
- Remote focus of the Z-Ax is via rotary knob on controller
- EPROM to store values. most settings now persist thru power down reset.
- Controller is configurable, most options are built in to the basic unit so less hardware options more parameters.
- No more confusing dip switches, all configuration is done with internal parameters and eprom.
- Serial Number and manufacture date in firmware. (see sernum, mandate)
- Higher voltage +24vdc supply, single +24VDC power supply. easier to obtain, lower cost,
- 3 axis encoder inputs. All controls have encoder inputs standard.
- internal or External joystick.
- Ludl Compatible command sets. All commands end/respond immediately, motion can then be monitored with the STATUS command. The position can be obtain with the where command during a motion.
- Fast speed. 24 revolutions / second or more than 480,000 pulses per second
- Smother motion => quieter. The low level wave forms have been optimized.
- An advanced sync algorithm for all motion. The motion will be in a vector and end at the same time.
- The most accurate speed control form 1 pulse per second to 480,000 pulses per second in integer steps.
- The most accurate acceleration profile and control including Jerk Control (acceleration profile control)
- The most accurate syncronization with external hardware. The controller comes standard with hardware/ firmware that can produce syncronization pulses accurate to a single microstep to syncronize external hardware. (see WSIZE, WINTERVAL WMODE)
- Backlash compensation (see Backlash Command)
- uni-directional approaching. An order of magnitude better repeatability with uni directional motion. (see UNIDIR command)
- More control over user inputs. (invert any joystick axis or z-encoder and enable/disable them individually)
- Better axis inverting. An axis can be inverted and all relative electronics are syncronized with it (limit switch, encoder direction....) => For example, neg limit switch bit is always the neg limit switch. if active you can only move in the positive direction.
- 600 nanosecond low level servo loop for very fast and stable encoder servoing.
- USB -> serial adapter built in. The control looks like a additional serial port.
- Better serial communication responces. Higher baud rate (57600) and the control now responds to serial requests within a few ms.

- Built in auto respond functions. The controller can be configured to automatically send a responce to certain situations (position updates, encoder Latching, End of Move EOM), thus avoiding time costly polling.
- seperate units control for the display and PC comunications. You can display one set of units on the display, while comunicating in differant units.
- New Power down mode. turns off motors and display, but can still comunicate over com port.
- Home no longer zero axis. Home can be to any limits switchs.
- Two coordinate systems, Machine coordinates and User Coordinates.
- Each axis has a built in serial number and eerpom for axis specific data.

CARE AND MAINTENANCE

Cleaning the painted or plastic surfaces.

Avoid the use of any organic solvents (such as thinner, alcohol, ether, etc.) to clean the painted or plastic surfaces of the accessory. Instead, use a mild solution of soap and water or a neutral detergent.

Never attempt to dismantle.

Never attempt to dismantle the instrument, thereby avoiding the possibility of impaired operational efficiency or accuracy. Contact an authorized Conix distributor for service and repair.

When not in use.

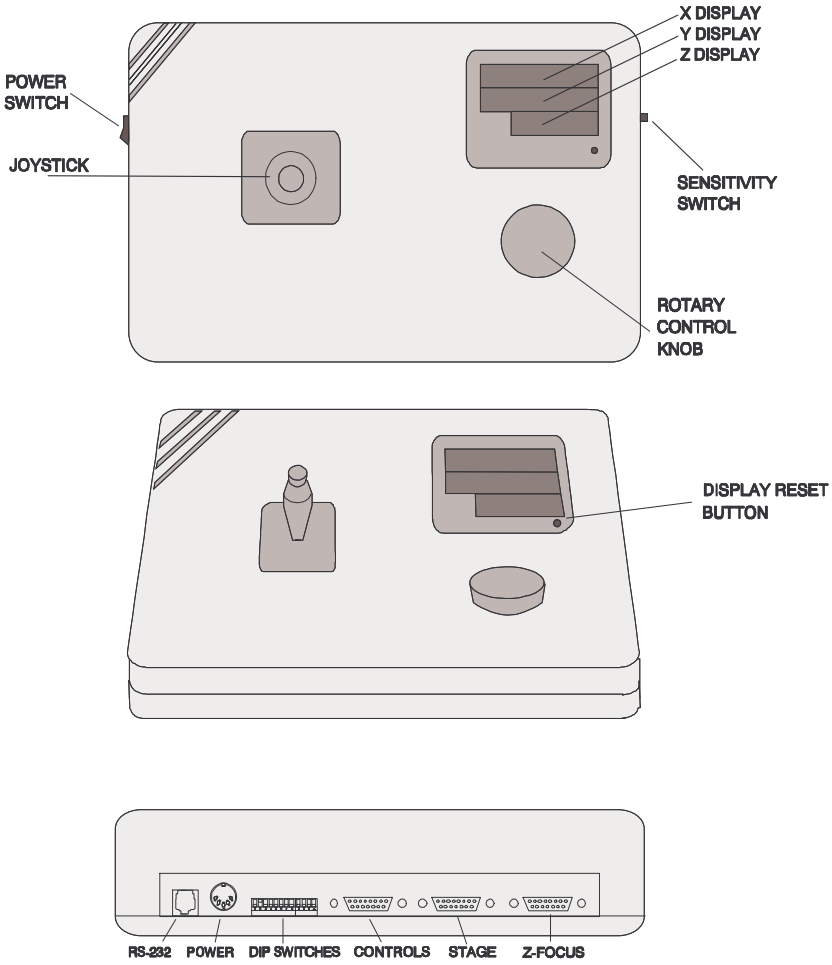
When not in use, turn off power to the accessory with the power switch on the Controller Module. When unit is not in use for an extended period, unplug transformer from its 120VAC 60Hz outlet.

Handle with care.

Handle all equipment with care. Install equipment in an environment with limited exposure to direct sun light, dust, high temperatures, humidity, and vibration. Avoid any machanical or electrical shocks.

NOTE: We reserve the right to make alterations in design or function. For this reason, specifications or illustrations in this manual may not conform with models in current production.

CONTROL UNIT



REAR VIEW OF CONTROLLER

INSTALLATION

Installation of Motorized Stage:

Install the stage onto your microscope stand. Hook up the stage cable by plugging it into back of the controller. Plug one end of the Z-Focus cable into the back of the controller and the other end into the Motor Mount Unit.

Check to see if stage hits the neck of the stand when pushed to the extreme rear limit. If it hits, adjust the rear limit stop so the stage engages the stop before hitting the neck. To adjust the limit stop, use the 0.05" allen wrench to loosen the set serew on the stop. Slide the stop to the desired position and retighten the set screw.

CAUTION: If a limit stop is moved too close to the outside edge of the stage, the physical limit may be reached before the limit stop is encountered. If that happens, a racheting noise will be heard and an error will occur. Readjust the limit stop in, toward the center of the stage, until limit stop is encountered.

Connect power cables.

Plug power supply cable into the round power connector on back panel of stage controller. Be sure to connect the power supply to the controller prior to plugging it into the wall and prior to turning on the unit. Plug the power supply into a 120VAC 60Hz receptacle.

Installation of the Motor Mount Unit

(1) Remove the fine focus knob that has the looking ring (may be on either side depending on microscope) from the microscope using the shorter of the two allen wrenches included in the parts package. Retain the spring washer on the fine focus shaft.

(2) Attach the pulley to the fine focus shaft with the pulley groove toward the microscope stand. Align pulley set screw to bear on the flat portion of the fine focus shaft. Use the longer of the two allen wrenches supplied in the parts package. Be sure to press the right fine focus knob and pulley towards each other while securing the pulley to compress the spring washer and ensure proper pulley position.

NOTE: Pulley must be installed with pulley groove towards the microscope stand for correct operation of the accessory.

(3) Attach the Motor Unit to the Coarse Tension Adjusting Ring of the coarse focus knob with the two thumb screws.

NOTE: Some models microscope stands have a small access plate near the Coarse Focus Knob, toward the rear of the stand. This access plate and screws must be removed as it will prevent the Motor Unit from mounting flat against the microscope stand.

NOTE: Make sure you plug the power connector into the control unit prior to plugging the supply into a wall receptacle.

OPERATION

Motorized Operation of the Stage.

1) Turn on Motorized Stage Controller by flipping power-on switch on left side of controller.
2) The 3-position Speed Select Switch (located on right side of controller) allows you to change the speed of the stage relative to the deflection of the joystick, slow, medium, fast.
3) Moving the joystick to the right moves the stage rightward, and moving the joystick to the left moves the stage leftward, etc. If the direction of the stage is opposite of what you desire, you can change the dip switch setting (refer to Dip Settings section of this manual) to invert the orientation of either the X-joystick or Y-joystick motion.
4) Pushing the joystick to an extreme position in one direction causes the stage to move quickly, while pushing gently on the joystick causes the stage to move relatively slowly. The speed of the stage is proportional to the deflection of the joystick.
5) Pressing the button on the joystick causes the stage to move many times faster. You can still control the speed by the amount of deflection from the center of the joystick.

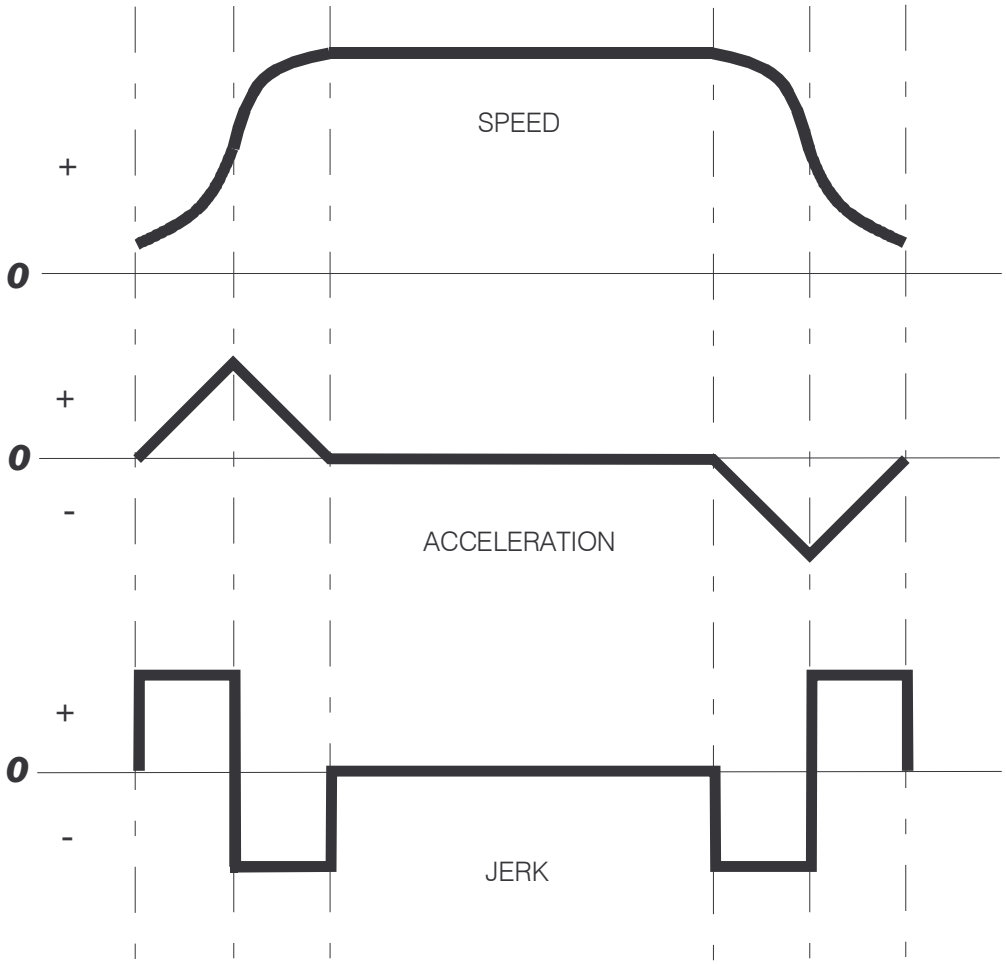
Ludl compatibility:

The Axisname are defaulted to XYZ but may be renamed using the **AXISNAME** command. The Axis ID for the Low-Level command set can be either X=1 or 24 , Y=2 or 25, Z= 3 or 26. The controller maps both address 1 and 24 to the X-Axis. This should be compatible with both the Ludl MAC 2000 and Ludl MAC 5000. Note: For ludl compatible mode, set '**COMUNITS UM1**' and '**DECIMAL OFF**' Unlike the Ludl command set, Our controllers do not repeat the last command when a <CR> is received without a command. The controller do not use Modul or Point Id's. Valid axis labels are dependent on the controller. An axis parameter without an assignment (=) is assumed to be an assignment of zero, unlike the Ludl command set which returns the current setting.

NOTE: On power up the computer senses the position of the joystick, this location becomes the 'dead zone'. Do not deflect the joystick during power up.

NOTE: On power up the computer senses the presence of the stage and will reconfigure itself to best match the stage if differnt then the last stage attached to the unit. The displays will flash all decimals on any axis that change on power up (new parameters). This should never occur in normal operation with the same stage attached.

SPEED, ACCELERATION, JERK



The Position, Speed, Acceleration and Jerk control is performed by an advanced high speed algorithm. Not only is the position controlled, but Speed (first derivative with respect to time), the Acceleration (the second derivative with respect to time) and the Jerk (the third derivative with respect to time). The advaced jerk control allows precise control over the force (Mass x Acceleration). The force is applied in an increasing manor that minimizes the "jerk" of the motion. The control produces the smoothes acceleration and deacceleration possible. The 'ACCEL' command actually sets the maximum acceleration as the speed sets the maximum speed. Note that the maximum speed or acceleration may not be obtain for any given motion.

COMPUTER CONTROL

Communication Specifications

The Stage controller Communications interface is a interface between a host computer and the controller. The communications is established through an USB connection configured as a RS-232C port, with 57600 baud, no parity, eight data bits, one stop bit, and hardware flow control(1). There are basically two methods to comunicate with the controller. The first method is a programming protocol with text (standard ASCII alpha-numeric characters), along with some special control characters such as carriage returns, spaces and tabs. The controller responds to a set of built-in commands with unique names. The commands can be executed by simply sending the command name with some parameters (if required). the controller will respond in ASCII and may include the result requested. The available commands and their functions are explained in Section I of this manual labeled 'High-Level Format'. This serial control feature can be accessed through terminal programs such as Telix, ProComm, and HyperTerminal. The command set mimics Ludl's command set, so software written with drivers for Ludl stages should be able to run the controller without modification 1. The Control command sequence also includes an abbreviated version of the commands that helps cut down on typing time and serial bus traffic.
There is also a 'Low-Level Format' described in Section II. With this format, the Interface is a simple transfer module. It will receive the device numbers, codes, and data from a host computer and transfer this information to appropriate modules. The characters used are in binary format. The host computer does most of the controlling and calculations. All necessary codes and their functions are explained in Section II.
Either format may be selected by software control. On power-up, the controller is in 'High-Level Format'. The Format can be changed by software by sending two binary bytes before sending the first command in the new format. For example, if the Interface is in High-Level Format, switch to Low-Level Format first, and then send the Low-Level commands to the Interface. Once switched to either format, the Interface will recognize only commands from that format and will stay in the same format until switched again.

NOTE1: The only hardware flow control that is implimented is the CTS (Clear To Send) is held inactive while a command is being processed. It is important to use hardware flow control when using Low-Level commands as there is no other means to wait for the controller to process the current command (some commands respond hence the pc waits for a responce, but others do not). It is possible to send a sequnce of commands that do not respond and over run internal buffers. As all High Level commands respond and no additional commands are transmitted while processing the last command, this is not a problem or requirement with High-Level commands.

RS 232 Timeout: There is a 2-second serial time out for low level commands. The High-Level commands have a ten second receive timeout.The controller will reset the internal buffers and wipe any partial data at the time out.

SERIAL DELAY: Due to the use of higher speed computers, there is no longer any need to delay serial communication replies; therefore, serial delays are not supported.

Note: There is no support for parity check.

Note: For any command that causes a reset (RESET, DEFAULTS, ... etc), no command should be sent for 3 seconds after the reset to allow for the system to come back up.

General Format Of Commands

Each line sent to the Controller should have a command, Data, and be terminated with a carriage return. The first item on the line should be the command. Each line can contain only one command and the Controller's commands are not case-insensitive. The allowed commands are listed below and a quick reference is included at the end of section I. After the command are the parameters, (Bracketed "[ ]" parameters are optional.) The axis name is given, followed immediately by an equal or sign and the axis parameter value. Each axis must be separated from the one before by one blank space. One or more axes may be specified on a single command line. An axis symbol typed without an "=" assignment is assumed to mean"=0", or the command format may not require a parameter value (e.g., **WHERE X**). Boolian expressions for axis commands are X+ or X-, for the X-Axis. Some boolean commands require an On/OFF parameter. Commands will accept bool (ON/OFF), integer or decimal numbers. Internal truncation or rounding will occur of fractional decimals that are beyond the significat digits of the particular axis/command. Standard Axis Names: X and Y are stage controls, Z is focus control. Axis Names maybe changed with the AxisName command. And finally, each command must be terminated with a command termination character specified here as <cr> has the value of 13 decimal, 0x0D hexadecimal and represented in C programming language by "\r" (carriage return). The carriage return indicates to the Controller the end of a command (=> process command). The entire command string cannot exceed 32 characters. Throught this manual we use axis names axis1=X, axis2=Y axis3=Z. This is just for ease and convineounce, the axis names can be reassigned to whatever character you like (see AXISNAME)

COMMAND [*axis=???***] [***axis=???***] [***axis=???***] <cr>**
COMMAND [*axis*[±]] [*axis*[±]] [*axis*[±]]**<cr>**
COMMAND [???]<cr>

All commands are completed with a Carriage Return (ASCII hex code: 0D). The controller receive ASCII characters one at a time and place them into a memory buffer. With the exception of single hex code commands like the tilde (~), the controller will not process a command in the memory buffer until the Carriage Return (<CR>) has been received. Some commands that cause state changes will be persistent even when power is removed and reapplied. These commands have a 'Persistent: Yes' in their discriptions.

MOVE X=1234<cr>
MOVE X=1234 Z=1234.5<cr>
MOVE X=1234 Y=1234 Z=1234<cr>
MOVE X Y Z<cr> (This is evaluated as **MOVE X=0 Y=0 Z=0**)
ENCODER X+ Y- Z-<cr> (X-Encoder on, YZ off)

RESPONSE

Valid Examples: (Typed commands are in **THIS TYPEFACE**; computer replies are in ***THIS TYPEFACE***.)

:A <DATA><EOL> Everything is ok <*returned data*>
:N <ERROR CODE><EOL> Error.

Every command returns a response: The response is in the form of a colon followed by a status character (either an A or N). The colon is sent by the Controller as soon as the command is received. The status character is not sent until the function has been processed. Do not send another command until the last function has been processed and returned a response. If for some unknown reason the Controller does not respond with a colon, then the command was not received properly (due to communications problems) and the command must be resent. In this case, the Controller's internal buffer must be emptied by sending an ESC (ASCII 27) or Backspace <bs>, <del> (ASCII 08) character . This is necessary since your last command may have been partially received and may still reside in the controller's internal buffer. The receive Buffer is purged if a command is not completed in 10 seconds for high level format and 2 seconds for low level format (same as ludl MAC). All results end with an End Of Line <eol> character or character sequence. The EOL character(s) can be set with the EOL command. The Default <eol> =<cr> (ASCII 13, 0x0D). Commands that cause motion or take an extended amount of time maybe monitored with the STATUS command. Some commands are prohibited while the status returns 'B'=BUSY. For example, you can not turn off motors while a move is in progress.

Examples:

Command:	**Where Z<cr> or W Z<cr>**	
Response:	***:A 1002<eol>***	
Command:	**ENCODER X+ Y- Z-**	(X-Encoder on, YZ off)
Response:	***:A X+ Y- Z-<eol>***	
Command:	**M Z=1001<cr>**	(move to location 1001)
Response:	***:A <eol>***	(everything is okay)
Command:	**W Z <cr>**	(where is z-axis?)
Response:	***:A 1001 <eol>***	(z-axis position is 1001)
command:	**AQRST<cr>**	(an illegal command)
response:	***:N -1 Unknown Command<eol>***	(error code -1)

PRESENTLY ASSIGNED ERROR CODES

When a command is received that the controller cannot interpret, for one reason or another, an error message is returned in the following format:

:N<error code>

The error codes are as follows:

-1 Unknown Command
-2 Unknown Axis (Unrecognized Axis Parameter [valid axes are dependent on the controller])
-3 Missing parameters (command received requires additional parameters such as x=1234)
-4 Value Out of Range (Parameter Out of Range)
-6 Undefined Error (command is incorrect, but for none of the above reasons)
-7 Power Down Error (power to one or more axis' is removed and cmd requires power)
-8 Axis not HOMEd yet. soft limits unavailable.
-21 Serial Command halted by the HALT command

Command: ACCEL

Shortcut:
Format: **ACCEL** [*Pulses*]
Default: 10000
Persistent: Yes
Function: This command sets the maximum acceleration of the stage in pulses / sec*sec. The value is an integer between 1 and 400000. It is the second derivative of position with respect to time. or the first derivative of speed with respect to time. It is the rate of change of speed with respect to time. since F=MA, this will set the maximum force that is applied to the stage as well. The control has a very advanced acceleration algorythm. The acceleration is ramped similaily to the speed. The acceleration starts at zero and is changed in units of **JERK** (see **JERK** and **SPEED**) to a peak or maximum **ACCEL**. then the acceleration is decrease back down to zero at the same time the maximum speed is obtained. This means that there is minimum forces at take off (startspeed) and at the approach of max **SPEED**. the process is reversed during the slow down or deacceleration on the approch to the final destination. The **SPEED** profile is like an S-Curve opposed t oa simple traposodal curve.

Example: **ACCEL 5000<cr>**
:A<eol>

See also **SPEED, JERK**

Command: AUTOCONFIG

Shortcut:
Format: **AUTOCONFIG** [*ON*][*OFF*]**<cr>**
Default: ON
Persistent: Yes
Function: This command will turn ON/OFF auto configuration of the stage or Z axis when a different stage is attached. The system uses the axis serial number (embedded in each axis) to determine if the axis has changed. The new axis must have a axis type the controller recognizes (see DEFAULTS for a list of axis types.) Upon a change, the system will load a set of defaults for the new axis type and then the parameters in the axis itself on top of those. This should effectively reconfig the controller for a new axis. If an axis does not have an embedded serial number, no detection of axis change and thus no parameter change will occur.
Reply: If there are no errors, the positive reply "***:A***" will be sent followed by the current state.
Example: **AUTOCONFIG OFF<cr>**
:A OFF<eol>
In this example, AutoConfig is turned off.
See also: **DEFAULTS, SERNUM**

Command: AUTOWHERE

Shortcut:

Format: **AUTOWHERE** [*axis*[±]] [*axis*[±]] [*axis*[±]]

Default: Power up default is X- Y- Z-

Persistent: No

Function: This command sets up the system to automatically send the where position, in binary, over the serial line every so oftem (about 13 times a second). You can select the axis you want to receive data on. The transfer is in binary and always starts with an axis byte (XWhereHexTx, YWhereHexTx, ZWhereHexTx), then 4 bytes of data LSByte first , ..., MSByte last. This function is active only in High-Level ASCII mode as it would interfere with low-level binary transfers. All ASCII bytes start with MSBit=0, all special binary transfers start with a byte that has MSBit=1. If no parameters are given the current state is retrieved.

XWhereHexTx, 0x81
YWhereHexTx, 0x82
ZWhereHexTx, 0x83

Reply: If there are no errors, a positive reply of "***:A***" will be followed by the current state

Example: **AUTOWHERE Z+<cr>**
:A X- Y- Z+<eol>
The command in this example will turn on the Z-Axis **AUTOWHERE** function. The controller will start sending '0x83 ZByte0, ZByte1, ZByte2, ZByte3 about 13 times a second. Byte0=Least significant byte, Byte3=Most significant byte.

See also: **ELATCHSEND**

Command: AXISINV

Shortcut:

Format: **AXISINV** [*axis*[±]] [*axis*[±]] [*axis*[±]]

Default: X- Y- Z-

Persistent: Yes

Function: This command will invert an axis. all the functions associated with the axis are reveresed (motor dir, limit switches, encoder dir). The only concern with using this command is the z-axis. If somebody inverts what up is and a piece of software thinks it is moving away from the sample, but is actually crashing into the sample. be careful

Reply: If there are no errors, a positive reply of "***:A***" will be followed by the current state

Example: **AXISINV X+<cr>**
:A X+ Y- Z-<eol>
The command in this example will invert the X-Axis.

Command: AXISNAME

Shortcut:

Format: **AXISNAME** [*new axis1*] [*new axis2*] [*new axis3*]

Default: X Y Z

Persistent: Yes

Function: This command sets or retrieves the names of the axis present. The new axis name can be any letter from A->Z. If you set a new name it is saved in the eprom so next time power is applied it will have the same name.

Example: **AXISNAME X Y B<cr>**
:A X Y B<eol>
The command in this example will rename the 3rd axis (normally Z) to B.

Command: AXISMEM

Shortcut:

Format: **AXISMEM** *axis [adddress[:value]] [write] [read]*

Default:

Function: This command will manipulate the internal memory and read/write it from/to the axis eeprom. There is an internal RAM that is a copy of data that may or maynot be an exact copy of what is in the axis eeprom. You can ensure the internal RAM is the same as the axis EEPROM by issuing a READ command. 'AXISMEM XREAD' for example. The Write command commites all the data to the axis EEPROM. Becareful you can cause a lot of problems if you set a value incorrectly. The address[:value] form of this command allows you to read/write the internal ram a 32-bit word at a time. The AXISREAD command is more efficient than this command as it operats on an entire block (axis) at a time. AXISWRITE operates on a entire block as well.

Example: **AXISMEM Z 0<cr>**
:A 00112233.<eol>
The command in this example will retrieve the data at byte offset 0 -> 3 (32 bits = 4 bytes)

See also: **AXISREAD, AXISWRITE, EPROMREAD, EPROMWRITE**

Command: AXISREAD

Shortcut:

Format: **AXISREAD** *axis#*

Default:

Function: This command will read the axis EEPROM, which if equiped, is located in the mechanism itself. XY are in the stage and z is in the z-axis motor mount. *axis* must be in the range of 1, 2, 3 for axis1, axis2, axis3. not the name of the axis, but the axis number is required. This command will send back data in an ascii hex format, every two bytes represents one byte of data. Maximum of 256 complete bytes of data will be transmitted. Note: AXISREAD data is NOT the same as the EPROMREAD data. AXISREAD is physically located on the axis mechanism, where as EPROMREAD data is stored in the controller. After all AXISREAD and AXISWRITE commands are completed, a RESET should be preformed to reestablish internal parameters.

Example: **AXISREAD 1<cr>**
:A 00112233445566...<eol>
The command in this example will request the Axis EEPROM data for the X-Axis.

See also: **AXISWRITE, EPROMREAD, EPROMWRITE**

Command: AXISWRITE

Shortcut:

Format: **AXISWRITE** *axis#*

Default:

Persistent: Yes

Function: This is the inverse to the AXISREAD command. This command will write to the axis EEPROM, which if equiped, is located in the axis mechanism itself. XY are in the stage and z is in the z-axis motor mount. *axis* must be in the range of 1, 2, 3 for axis1, axis2, axis3. not the name of the axis, but the axis number is required. This command will send back the amount of data the controller expects. you must then send the data in binary. Note: this violates the standard communication specification. The controller will wait until it receives the data. Maximum of 256 bytes of data will be required. Note: AXISWRITE data is NOT the same as the EPROMWRITE data. AXISWRITE is physically located on the axis mechanism, where as EPROMWRITE data is stored in the controller. Send only the amount of data the controller requests. After all AXISREAD and AXISWRITE commands are completed, a RESET should be preformed to reestablish internal parameters.

Example: **AXISWRITE 2<cr>**
:A 256.<eol>
Send 256 bytes of binary data.
The command in this example will prepare the controller to receive the Y-Axis Axis EEPROM data

See also: **AXISREAD, EPROMREAD, EPROMWRITE**

Command: BACKLASH

Shortcut: **B**
Format: **BACKLASH** [*axis=steps*] [*axis=steps*] [*axis=steps*]
Default X=0, Y=0, Z=0
Persistent: Yes
Function: This command sets (or displays) the amount of backlash compansation (in units of stepper motor microsteps. 40000 steps / rotation). The value is an integer between 0 and 65535. This value is used to prewind the motors a given amount whenever the direction of the motor changes. If the value is too small (ie the system has more backlash then is compensated), when the motors change direction, there will be motor motion without any stage motion. If this number is too large, when the direction is changed the stage will hop forward into the new direction.

Example: **B X=5 Y=0 Z=0**
:A
The command in this example will make the X-Axis move 5 microstep in the new direction everytime the direction is changed.

See also: **UNIDIR**

Command: BUTTONS

Shortcut:
Format: **BUTTONS**[*ON*][*OFF*]
Default On
Persistent: Yes
Function: This command will retrieve or set the buttons bit. This will enable / disable user control buttons. This bit is used for all buttons except axis control buttons (use joystick +- to enable and disable axis controls). For example, this command will disable the zero button.
Reply: If there are no errors, the positive reply "***:A***" will be sent followed by the current state.

Example: **BUTTONS OFF**
:A OFF<eol>
In this example, the BUTTONS are disabled. The zero button no longer will cause the system to zero.

See also: **JOYSTICK**

Command: COMUNITS (communication Units)

Shortcut:

Format: **COMUNITS** [*MM*][*UM*][*UM1*][*UM01*][*NM*][*INCH*]

Default: MM

Persistent: Yes

Function: This command will Get / Set the current units that are used with communications between the PC and controller. The units can be set to

MM = millimeters 1mm = 1.000 (binary numbers 1 = 1um)
UM = Microns 1um = 1. (binary numbers 1 = 1um)
UM1 = Microns Tenths 1um = 1.0 (binary numbers 1=0.1um)
UM01 = Microns Hundreths 1um = 1.00(binary numbers 1=0.01um)
NM = Nanometers. 1nm =1. (binary numbers 1 = 1nm)
INCH = inches, 1" = 1.0000 (binary numbers 1 = 0.0001")

Reply: If there are no errors a positive response is sent back with the current setting.

Example: **COMUNITS MM<cr>** The comunits are changed to millimeters.
:A MM<eol>

Example: **COMUNITS INCH<cr>** The comunits are changed to inches.
:A INCH<eol>

See also: **UNITS (display units), DECIMAL**

NOTE: All of the commands that accept or return values in real units use the current setting of COMUNITS for scaling.

Note: Switching to low level mode will force this value to UM1 or UM01

Command: DECIMAL

Shortcut:

Format: **DECIMAL** [*ON*][*OFF*]**<cr>**

Default: ON

Persistent: Yes

Function: This command will turn ON/OFF decimal point transmitting. When ON, the commands that quary positional data may include a decimal point and significate digits to the right of the decimal. If OFF all the digits to the right of the decimal are rounded to an integer and the integer is returned. Even if decimal is turned OFF, the control will still accept decimals, it just won't transmit them.

Reply: If there are no errors, the positive reply "***:A***" will be sent followed by the current state.

Example: **DECIMAL OFF<cr>**
:A OFF<eol>
In this example, decimal point transmition is turned off. This maybe requied to be compatible with some software.

See also: **WHERE**

Command: DEFAULTS

Shortcut:

Format: **DEFAULTS** *[RP][BS][RPTheta][BSTheta] or [Axis1_type# Axis2_type# Axis3_type #]*

Persistent: Changes Controller Eprom.

Function: This command reestablishs a set of standard defaults parameters, wiping out all current eprom settings. This command will end with a reset and reload the new set of parameters. With no parameters, the command reads the axis data and establish axis type and datum from the axis eproms. If no eproms are found, the behaior defaults to X=BS, Y=BS, Z=FZ types. RP = Rack and Pinion stage, BS=BallScrew stage, RPTheta= Rack and pinion Rotary (Theta) stage, BSTheta = BallScrew Rotary (Theta) stage. The alternative form of this command allows the user to specify each axis type individually. This command may take a second or two to set the defaults and then some additional time for the RESET.

Axis Types #:

1 = BS Axis Type
2 = LS Axis Type
3 = RP Axis Type
4 = Rotary (Theta) Axis Type
5 = FZ axis type Z - Focus Axis type (40um / Motor Rotation, 100um fine focus)
6 = Z - Focus Axis type (50um / Motor Rotation)
7 = Z - Focus Axis type (80um / Motor Rotation, 200um fine focus)

The standard BS stage system is equivalent to

Axis1_Type = 1, Axis2_Type = 1, Axis3_Type = 5

DEFAULTS BS <=> DEFAULTS 1 1 5

Reply: If there are no errors, a positive reply of "***:A***" will be sent.

Example: **DEFAULTS BS<cr>**

:A<eol> (long delay after :A to set data and reset control)

See also: **RESET**

Command: EINDEX

Shortcut:

Format: **EINDEX** [*axis*[±]] [*axis*[±]] [*axis*[±]]

Default: X- Y- Z-

Persistent: Yes

Function: This command is used for motor drivers with encoders installed. It actives a mode where by an active index line (XYZ) will cause the current encoder position to be overriden by the value in **EINDEXPOS** (see also **EINDEXPOS**). Normally an active index will only cause the encoder position to be captured into the latch buffer, but otherwise does not effect the current click counts. With **EINDEX** activated, this will cause the current encoder clicks to be wiped out during an active index pulse and replaced by **EINDEXPOS**. The capture into the latch buffer occurs before the over writting of encoder clicks, so it is possible to retrieve the **ELATCH** position and see where the index took place. This command does not require the **ENCODER** command to be active.

Reply: If there are no errors, a positive reply of "***:A***" will be followed by the current state

Example: **ENCODER Z+<cr>**

:A X- Y- Z+<eol>

The command in this example will turn on the Z-Axis encoder indexing. The next time the ZIndex line goes active (low) then the z encoder position will be preloaded with the Z **EINDEXPOS**

See also: **EWHERE**

Command: EINDEXPOS

Shortcut:
Format: **EINDEXPOS** [*axis=position*] [*axis=position*] [*axis=position*]
Default: X=0 Y=0 Z=0
Persistent: Yes
Function: This command is used for motor drivers with encoders installed. This is the value that is preloaded when the **EINDEX** function is active and an index signal is present. This command does not require the **ENCODER** command to be active.
Reply: If there are no errors, a positive reply of "***:A***" will be followed by the current state

Example: **EINDEXPOS Z=1000<cr>**
:A 0.0 0.0 100.0<eol>
The command in this example will set the **EINDEXPOS** for the z-axis to 100.0 (in COMUNITS).
See also: **EINDEX**, **EWHERE**

Command: ENCODER

Shortcut:
Format: **ENCODER** [*axis*[±]] [*axis*[±]] [*axis*[±]]
Default: X- Y- Z-
Persistent: Yes
Function: This command is used for motor drivers with encoders installed. This command tells the system to use the encoder inputs for positional information. Normally positional information is obtained from stepper motor steps, but in this mode the information from the encoders overrides the step count data. You can observe the numbers changing on the display when you physically move the stage with this command on. Even if this command is off for a perticular axis the encoder data canbe viewed using the **EWHERE** command.
Reply: If there are no errors, a positive reply of "***:A***" will be followed by the current state
Example: **ENCODER Z+<cr>**
:A X- Y- Z+<eol>
The command in this example will turn on the Z-Axis encoder for positional data.
See also: **EWHERE**

Command: ENCODERNM

Shortcut:
Format: **ENCODERNM** [*axis=nm*] [*axis=nm*] [*axis=nm*]
Default: X=1000 Y=1000 Z=500
Persistent: Yes
Function: This command is used for motor drivers with encoders in stalled. This command is used to scale the encoder clicks to real units. Enter the encoder click size in nanometers. Range {Min=0, Max=65535}
Reply: If there are no errors, a positive reply of "***:A***" will be followed by the current state
Example: **ENCODER Z=100<cr>**
:A 1000 1000 100<eol>
The command in this example will change the Z encoder scale to 100nm per encoder click.
See also: **EWHERE**

Command: EHOLD

Shortcut:
Format: **EHOLD** [*axis*[±]] [*axis*[±]] [*axis*[±]]
Default: Power up defaults is X- Y- Z-
Persistent: No
Function: This command is used for motor drivers with encoders installed. This command tells the system to Hold/freeze an encoder axis input. Any encoder clicks occuring during a hold will be ignored.
Reply: If there are no errors, a positive reply of "***:A***" will be followed by the current state

Example: **ENCODER Z+<cr>**
:A X- Y- Z+<eol>
The command in this example will hold the current Z-Axis encoder position.
See also: **ENCODER, EWHERE**

Command: ELATCH

Shortcut: **EL**
Format: **ELATCH** [*axis*] [*axis*] [*axis*]
Default: Power up default is 0 0 0
Persistent: No
Function: This command is used for motor drivers with encoders installed. This command is used to retrieve the current latch buffer. This is the value/position of the encoder when the last Index bit was activated. If no axis is given all axis are returned.
Reply: If there are no errors, a positive reply of "***:A***" will be followed by the current state

Example: **ELATCH Z<cr>**
:A 1.0<eol>
The command in this example fetched the encoder latch buffer and it was latched at position 1.0 (in COMUNITS).

See also: **ELATCHSEND**

Command: ELATCHSEND

Shortcut:

Format: **ELATCHSEND** [*axis*[±]] [*axis*[±]] [*axis*[±]]

Default: Power up default is X- Y- Z-

Persistent: No

Function: This command sets up the system to automatically send the encoder latch postion whenever a specific encoder index becomes active. The Encoder XYZIndex pins on the Controls connector cause the encoder system to latch (Capture the current encoder position into a latch buffer). This command then transfers the latch buffer data to the PC asycronously in binary fomat. The Latch of the encoder position occurs even if this command is not active, so the data could also be monitored by polling the **ELATCH** buffer. The transfer is in binary and always starts with an axis byte (XELatchHexTx, YELatchHexTx, ZELatchHexTx), then 4 bytes of data LSByte first , ..., MSByte last. This function is active only in High-Level ASCII mode as it would interfere with low-level binary transfers. All ASCII bytes start with MSBit=0, all special binary transfers start with a byte that has MSBit=1.

```
XELatchHexTx,    0x84
YELatchHexTx,    0x85
ZELatchHexTx,    0x86
```

Reply: If there are no errors, a positive reply of "***:A***" will be followed by the current state

Example: **ELATCHSEND Z+<cr>**

:A X- Y- Z+<eol>

The command in this example will turn on the Z-Axis **ELATCHSEND** function. Eveytime the Z-Index becomes active the system will latch (capture the encoder position into the latch buffer) and send it to the PC (in COMUNITS). The Data stream should look like '0x86 ZByte0, ZByte1, ZByte2, ZByte3'

See also: **AUTOWHERE**

Command: EOL

Shortcut:

Format: **EOL**[*nn*][*mm*]**<cr>**

Default: **EOL=0D=<cr>**

Persistent: Yes

Function: This command allows you to set / retrieve the end of line EOL byte or bytes. nn and mm are hexadecimal numbers. for example 0D/<cr> or 0a/<n> or a combination 0D0A <cr><n> to set a new EOL sequence.

Reply: If there are no errors, the positive reply "***:A***" will be sent followed by the current state.

Example: **EOL<cr>**

:A OD<eol>

In this example, the EOL is 0x0D.

Command: EOM

Shortcut:

Format: **EOM** [*ON*][*OFF*]

Default: Power up default OFF

Persistent: No

Function: This command will retrieve or set the End of Move (**EOM**) sending. If ON a single byte EndOfMoveHexTx=0x80 will be sent when the motion has completed . The function is only valid while in High-Level ASCII mode.

Reply: If there are no errors, the positive reply "***:A***" will be sent followed by the current state.

Example: **EOM ON**

:A ON<eol>

In this example, the **EOM** sending is turned on. At the completion of all commanded moves the byte 0x80 will be sent to the PC.

Command: EPROM (CAUTION)

Shortcut:

Format: **EPROM***Address*[*:Value*]

Persistent: Yes (you're changing the Eprom data)

Function: This command will retrieve or set an eprom value at a specific address. The address is the byte offset within the eprom memory, the value returned or set is 4 bytes long consisting of byte address to byte address + 3. Becareful with this command as it is possible to really cause server problems if you set a value incorrectly. The EPROM is used to store all parameters that are persistant after a power cycle. This command is used mainly to back up the parameters and restore them. It is more efficient to use the EPROMREAD and EPROMWRITE commands, as they operator on a block of data at a time.

Reply: If there are no errors, the positive reply "***:A***" will be sent followed by the eprom value.

Example: **EPROM 10<cr>**

:A 10: 00112233<eol>

See also: **AXISREAD,AXISWRITE,EPROMREAD,EPROMWRITE**

Command: EPROMREAD

Shortcut:

Format: **EPROMREAD** *Axis#*

Function: This command will retrieve the block of data for the given axis. Axis is in the range of 0, 1, 2, 3 where 0=base data, 1=Axis1, 2=axis2, 3=axis3. The data is sent back in hex format AABBCC.....<eol>. every two characters is a byte of data. The block size will not exceed 256 complete bytes.

Reply: If there are no errors, the positive reply "***:A***" will be sent followed by the data in ascii hex format

Example: **EPROMSIZE 1**

:A 00112233445566778899...<eol>

In this example, Axis1 (X-Axis) eprom data is sent back.

See also: **AXISREAD, AXISWRITE, EPROMWRITE**

Command: EPROMWRITE (CAUTION)

Shortcut:

Format: **EPROMWRITE***Axis#*

Persistent: Yes

Function: This command will set a block of EEPROM data into the controller for the given axis. Axis is in the range of 0, 1, 2, 3 where 0=base data, 1=Axis1, 2=axis2, 3=axis3. The responce will indicate the number of binary data bytes it expects you to send. This should be the same size as the EPROMRead command for the given axis. NOTE: this violates the standard communication protocol. you must send x bytes of binary data immediately after this command. The controller will wait indefinately for the data.

Reply: If there are no errors, the positive reply "***:A***" will be sent followed by the epromsize.

Example: **EPROMSIZE 3**

:A 256<eol>

Send 256 bytes of binary data.

In this example, Axis3 (Z-Axis) eprom data is waiting to be received by the controller..

See also: **AXISREAD,AXISWRITE,EPROMREAD**

Command: EPULSES

Shortcut: **EP**
Format: **EPULSES**
Function: This command is used for motor drivers with encoders installed. This command returns the current encoder pulse count for all axes. This value is raw and not converted into any units
Reply: If there are no errors, a positive reply of "***:A***" will be followed by the current position

Example: **EP X Y Z**
:A 100 200 300

See also: **ENCODER, EWHERE**

Command: EWHERE

Shortcut: **EW**
Format: **EWHERE** [*axis*] [*axis*] [*axis*]
Function: This command is used for motor drivers with encoders installed. This command returns the current encoder position of the motor for the axis specified. (if blank all axis' are returned) in **COMUNITS** units
Reply: If there are no errors, a positive reply of "***:A***" will be followed by the current position, in current **COMUNITS** (MM, UM, UM1, UM01, NM, INCH)
For this example assume: X=1.234567mm Y=7.654321mm Z=0mm from the origin.

Example: **EW X Y Z**
:A 1.234567 7.654321 0.0 (with **COMUNITS**=MM and **DECIMAL** ON)

See also: **ENCODER, WHERE, COMUNITS, DECIMAL.**
Note: the order of the X, Y, and Z's specified in the **EWHERE** command, is the order in which they are returned. The reporting precision of the **EWHERE** command can be changed with the **COMUNITS** command and the Setup Control Commands (below). If fractional decimals cannot be handled by the user's software, use the appropriate **COMUNITS** (UM1=> 1=100nanometer) and set **DECIMAL**=OFF.

Command: HALT

Shortcut \(the backslash character)
Format: **HALT**
Function: This command will stop all active motors.
Reply: If there are no errors, a positive reply of "***:A***" will be returned. If the "**HALT**" command is given while a commanded move is *in motion*, the controller will reply with the ***:N-21*** error.

Example: **HALT**
:A

Command: HERE

Shortcut: **H**

Format: **HERE** axis=*position* [axis=*position*] [axis=*position*]

Default: Power up default is 0, 0, 0

Function: Assign the specified number to the axis's current position buffer. The unit of measurement is in **COMUNITS** (see **COMUNITS**). This defines the current position to be a specific distance from the origin (0), i.e., the origin may change.

Reply: If there are no errors, the positive reply "***:A***" will be sent back from the controller.

Example: **H X=10 Y=43.21 Z**
:A<eol>
If the **COMUNITS** is mm, then the X position will change to 10.0mm from the origin, Y will change to 43.21mm, and the Z will be zeroed. The display mmediately shows the new values (The display is governed by **UNITS** which maybe different from **COMUNITS**)

Command: HOME

Shortcut: **!** (the exclamation point character)

Format: **HOME** [*axis*[±]] [*axis*[±]] [*axis*[±]]

Function: Move specified axis motors toward their physical limit switch. When the limit switch is reached, the motors turn around and move away from the switches a distance of HOMEDIS, The motors then reapproach the limits at a slow speed. This greatly improves the accuracy of the home position and is independant of the system speed. The completion of the home command causes the internal machine coordinate system to be preset with either the STROKEMINUS or STROKEPLUS values (see MACHINEWHERE). The HOME command uses the LIMITSW+- bits to determine if limit sensors exist. If the limit sensor exists in the direction requested then the home is performed normally. If the limit sensors are not present, there is no motion, but the internal machine coordinate is preset with the STROKEPLUS or STROKEMINUS and with SOFTLIMITS enabled only motion in the opposite direction is permitted. Note: The current position buffer is not zeroed or changed; use the **HERE** or **ZERO** command to set a specific axis to zero, if desired. Whether the move is toward the upper or lower limit switch depends on the [±]. The default motion is for X- Y- (Home the XY axis only toward their negative limit switches), but this can be overridden with X+ Y+ and send the stage to the upper limits instead. you can also include Z, but if no limits are present only the presetting of the machine coordinates are performed. if at all possible, it is a good idea to move z-axis up first, thus avoiding any potentional interferances, then **HOME** the XY stage. If SOFTLIMITS are enabled, they will be active after this command. If the STROKEMINUS or STROKEPLUS are changed another home must be performed for the new values to take effect.

Reply: A positive reply of "***:A***" is sent back when the command is received correctly. Reception of the reply does not mean the end of execution, and the command **STATUS** can be used to determine if the moves have been completed.

Example: **Home<cr>**
:A<eol>
The X and Y axis motors will start moving towards their lower limits switches. A **HALT** command can stop the motors.

Example: **Home X Y+<cr>**
:A<eol>
The X axis will home to its negative limit, the Y axis motor will home to its positive limit switch. A **HALT** command can stop the motors.

Example: **Home X+<cr>**
:A<eol>
The X axis will home to its positive limit. A **HALT** command can stop the motors.

Example: **Home Z-<cr>**
:A<eol>
Assuming no limits sensors with the z-axis and SOFTLIMITS Z+ (enabled), the z axis machine coordinate will be present to the Z-STROKEMINUS value and only motion in the Z+ direction will be permitted (up to the STROKEPLUS position). To move beyond the STROKE limits, SOFTLIMITS must be disabled

Note: The stage will be positioned at the limit switches at the completion of this command. Only subsequent commands to move the stage away from the limits will cause motion.

See also: **STATUS**

Command: HOMEDIS

Shortcut:

Format: **HOMEDIS** [*axis=distance*] [*axis=distance*] [*axis=distance*]

Persistent: Yes

Function: This will specify the amount of motion away from the home sensor for more accurate positioning.

Reply: A positive reply of "***:A***" is sent back with the current settings.

Example: **HomeDis Y=0.5<cr>**
:A<eol>
The X and Y axis motors will start moving towards their lower limits switches. A **HALT** command can stop the motors.

See also: **HOME**

Command: INBIT

Shortcut:

Format: **INBIT1**
INBIT2
INBIT3

Function: This command will get the state of the specific input bit. Input bits are shared with (XYZ) index pins on the controls connector. These Bits are active low. ON=0vdc, OFF=5VDC. You should not drive these pins if you are using the windowing mode (see WMODE, WINTERVAL, WSIZE) as this mode will drive the bits.

Reply: If there are no errors, the positive reply "***:A***" will be sent followed by the current state.

Example: **INBIT1<cr>**
:A ON<eol>

Command: JERK

Shortcut:
Format: **JERK** [*Pulses*]
Default: 250
Persistent: Yes
Function: This command sets the **JERK** (const value) in pulses / SecSecSec. The value is an integer between 1 and 30,000. It is the third derivative of position with respect to time. or the second derivative of speed with respect to time. and the first derivative of acceleration. It is the rate of change of acceleration with respect to time. It sets the slope or ramp of the acceleration, thus controlling the ramp of the forces applied to the stage and specimen. The control has a very advanced position, **SPEED, ACCEL, JERK** algorithm. The acceleration is ramped similaily to the speed. The acceleration starts at zero and is changed in units of **JERK** (see **ACCEL** and **SPEED**) to a peak or maximum **ACCEL**. then the acceleration is decrease (-**JERK**) back down to zero at the same time the maximum speed is obtained. This means that there is minimum forces at take off (startspeed) and at the approach of max **SPEED**. the process is reversed during the slow down or deacceleration on the approch to the final destination. The **SPEED** profile is like an S-Curve opposed t oa simple traposodal curve.
Reply: If there are no errors, the positive reply "***:A***" will be sent followed by the current state.
Example: **JERK 250<cr>**
:A 250<eol>
See also: **WHERE, SPEED, ACCEL**

Command: JOYSTICK

Shortcut: **J**
Format: **JOYSTICK** [X±] [Y±] [Z±]
Default: X+ Y+ Z+
Persistent: Yes
Function: This command enables (+) or disables (–) the input from the default manual control device for the axis (joystick, knob, buttons). The command state is remembered after power down.
Reply: If there are no errors, the positive reply "***:A***" will be sent followed by the current state.
Example: **J X+ Y+ Z-<cr>**
:A X+ Y+ Z-<eol>
The above command enables the default X and Y joystick control and disables the Z control knob.

Command: JSINV

Shortcut:
Format: **JSINV** [*axis*[±]] [*axis*[±]] [*axis*[±]]
Default: X- Y- Z-
Persistent: Yes
Function: This function will invert the XY joystick or Z rotary encoder input and Z-buttons. This allows for the invertion of user controls.
Reply: If there are no errors, a positive reply of "***:A***" will be followed by the current state
Example: **JSINV X+ Z+<cr>**
:A X+ Y- Z+<eol>
The command in this example will invert the X-Axis joystick control and the Z-Axis rotary encoder control

Command: JSPOS

Shortcut:
Format: **JSPOS**
Default: By sampling power up default is 0, 0
Function: This command will quary the current joystick position. It returns a value between -100 and 100, with 0 being at rest. On power up the controller samples the joystick position and this becomes the rest position. any deflection from this location is the jspos.
Reply: If there are no errors, a positive reply of "***:A***" will be followed by the current state

Example: **JSPOS<cr>**
:A 25 -100<eol>
The current position (x,y) of the joystick.

Command: JSROTSCALE

Shortcut:
Format: **JSROTSCALE** [*new scale*]
Default: 2
Persistent: Yes
Function: This command changes the scale of the Z rotary encoder user input. Larger number means each z rotational encoder click will produce a large motion. The new scale value can be 1 to 20.
Reply: If there are no errors, a positive response of "***:A***" will be sent followed by the current state.

Example **JSROTSCALE 4<cr>**
:A 4<eol>
In this example the **JSROTSCALE** will be change to 4. Each encoder click starts with 4 as the distance to travel.
Note: the side slide switch as cause this number to be scaled.

Command: JSSPD

Shortcut: **JS**
Format: **JSSPD** *axis=speed* [*axis=speed*] [*axis=speed*]
Persistent: Yes
Function: This command sets the relative motor speed for maximum deflection of the joystick to the value specified. (see **SPEED**) Speed is set in **COMUNITS** per second. The side slide switch allows for 3 addition 1 1/2 1/4 of this value settings. The turbo button allows for a speed of 1/2 the **SPEED** command value for full deflection.
Reply: If there are no errors, the positive reply "***:A***" will be sent back from the controller. The query reply tells you what the current JSSPD is

Example: **JS 10000**
:A 10000<eol>
(10,000 pulses per second max joystick speed)

Command: LIMITS

Shortcut:

Format: **LIMITS**

Function: This command gets the current limits state. The value returned as an ASCII number should be interpreted as binary number that represents each limit switch bit as follows:
Bit 0 = X positive Limit
Bit 1 = X negative Limit
Bit 2 = Y positive Limit
Bit 3 = Y negative Limit
Bit 4 = Z positive Limit
Bit 5 = Z negative Limit
Only motion away from an active limit switch will be successful.

Reply: If there are no errors, the positive reply "***:A***" will be sent back from the controller with the current limits switch settings.

Example: **LIMITS<cr>**
:A 10
In this example, the return value 10dec = 00001010 binary => Byte.1=1 and Byte.3=1. This means the X and Y lower limits are active. Only XY positive motion will be realized.

Command: LOWI

Shortcut:

Format: **LOWI** [*axis*±] [*axis*±] [*axis*±]

Persistent: Yes

Function: This command enables (+) or disables (-) low currunt (I)

Reply: If there are no errors, the positive reply "***:A***" will be sent back from the controller.

Example: **LOWIX-Y-Z+**
:A
This example turns low current on in Z-axis only.

See also: **POWER**

Command: MANDATE

Shortcut:

Format: **MANDATE<cr>**

Persistent: Yes

Function: This command will quary the system for the controller serial number.

Reply: If there are no errors, the positive reply "***:A***" will be sent followed by the serial number.

Example: **MANDATE<cr>**
:A 123<eol>

See also: **SERNUM**

Command: MOTCTRL

Shortcut: **MC**
Format: **MOTOCTRL** [*axis*±] [*axis*±] [*axis*±]
Persistent: No
Function: This command enables (+) or disables (-) the controller's ability to control the motor of a certain axis. The motor control voltage is set to zero. The electronics of the controller will effectively disengage the motor, however, it should be noted that there will still be a small detent force assoctiated with the stepper motors. If equipped with encoders, they can still be read and monitored for positional information.
Reply: If there are no errors, the positive reply "***:A***" will be sent back from the controller.

Example: **MC X+ Y+ Z-**
:A
This example shows that the X and Y motor control is enabled, but disables the Z motor control.
See also: **POWER**

Command: MOVE

Shortcut: **M**
Format: **MOVE** axis=*position* [axis=*position*] [axis=*position*]
Default: power up default is 0, 0, 0
Function: Move one or more axis motors to an absolute position. The unit of measurement is in **COMUNITS**. If no position is specified, a position of 0 (the origin) is assumed.
Reply: A positive reply of "***:A***" is sent back when the command is received correctly. Reception of the reply does not mean the end of execution, and the command **STATUS** can be used to determine if the move has been completed.

Examples: **M X=1234 Y=4321 Z**
:A
The controller will move the X-axis to position 1234 units from the origin using the maximum set speed (see **SPEED**). Simultaneously, it will move the Yaxis to position 4321 units, and the Z-axis to the zero (0) position. During this movement, the Joystick and Encoder inputs will be locked-out and cannot alter the target positions entered. The motors will stop when they have reached their target or when their limit switch is encountered. To stop the motors during a serial **MOVE** command, use the **HALT** (\ ) command.
See also: **STATUS, HALT**

Command: MOVREL

Shortcut: **R**

Format: **MOVREL** axis=*distance* [axis=*distance*] [axis=*distance*]

Function: Move one or more axis motor a distance relative from its current position. This command is very similar to the **MOVE** command. The unit of measurement is also in current **COMUNITS** (see **COMUNITS**)

Reply: A positive reply of "***:A***" is sent back when the command is received correctly. Reception of the reply does not mean the end of execution, and the command **STATUS** can be used to determine if the move has been completed.

Examples: **R X=1234 Y=-321 Z**

:A<eol>

The controller will move the X-axis a relative 1234 units in the positive direction at the maximum set speed (see **SPEED**). Simultaneously, the Y-axis will move 321 units in the negative direction, while the Z-axis will not move at all. During this movement, the Joystick and Encoder input will be locked-out and cannot alter the target positions entered. The motors will stop when they have reached their target, or if their limit switch is encountered. To stop the motors during a serial **MOVREL** command, use the **HALT** (\) command.

See also: **STATUS, HALT**

Command: MWHERE

Shortcut: **MW**

Format: **MWHERE** [*axis*] [*axis*] [*axis*]

Default: Not valid until after a HOME.

Function: This command is used to get the machine coordinate system. This is the coordinate system fixed relative to a limit sensor. A home must be preformed before this value is valid. Once a home has been completed for a given axis, this corrdinate system does not change even if you issue a HERE or ZERO command and change the origin.

Reply: If there are no errors, a positive reply of "***:A***" will be followed by the current state

Example: **MWHERE X<cr>**

:A 10.0 <eol>

The command retrieves the X axis machine coordinate. Note the units are in COMUNITS.

Command: NMPERREV

Shortcut:

Format: **NMPERREV** [*axis*=] [*axis*=] [*axis*=]

Default: X=2000000 Y=2000000 Z=40000

Persistent: Yes

Function: This command is used for linear axis' to scale the stepper motor rotation into linear motion in nanometers.

Reply: If there are no errors, a positive reply of "***:A***" will be followed by the current state

Example: **NMPERREV X=1000000<cr>**

:A 1000000 2000000 40000<eol>

The command in this example will change the nanometers per rev of the x axis to 1,000,000nm=1mm. this is the correct setting for a x axis ballscrew with 1mm pitch. This value is used to the motor steps into real units

Command: NOIDDELAY

Shortcut:
Format: **NOIDDELAY** (Requires additional Solenoid hardware)
Default: 3000
Persistent: Yes
Function: This command is for controllers equipped with a solenoid driver. The function sets the amount of time in milliseconds the solenoid is active for.
Reply: If there are no errors, a positive reply of "***:A***" will be followed by the current state

Example: **NOIDDELAY 1000 <cr>**
:A 1000<eol>
In this example the solenoid on time is set to 100ms = 1 second.
See also: **SOLENOID**

Command: ORIGIN

Shortcut:
Format: **ORIGIN** [*axis*] [*axis*] [*axis*]
Function: This command will retrieve the position of the origin of the user corrdiante system relative to the Machine Coordinate system. This value changes when a ZERO or HERE cammand changes the user coordinate system. Note units are in COMUNITS
Reply: If there are no errors, the positive reply "***:A***" will be sent followed by the current state.

Example: **ORIGIN X<cr>**
:A 9.56<eol>
The current origin of the user coordinate system relative to the machine coordinate system is 9.56 in the x axis (units as per COMUNITS)
See also: **MWHERE, WHERE, HERE, ZERO**

Command: OUTBIT

Shortcut:
Format: **OUTBIT1** [*ON*][*OFF*]
OUTBIT2 [*ON*][*OFF*]
OUTBIT3[*ON*][*OFF*]
Default: Power on default is ON => 0VDC on output pin
Function: This command will set or retrieve the state of the output bits (CONTROL port pins). These Bits are active low. ON=0vdc, OFF=5VDC. These pins have a limited drive capability. There should only be 1 TTL load per pin.
Reply: If there are no errors, the positive reply "***:A***" will be sent followed by the current state.

Example: **OUTBIT1 ON<cr>**
:A ON<eol>

Command: POWER

Shortcut:

Format: **POWER** [*ON*][*OFF*]**<cr>**

Function: This command will turn power On or OFF. Power on/off only turns power off to the display and motors. The main power is still applied and you can still comunicate with the control. If you try to move and axis you get a power off error. Note that **POWER** ON does not generate a reset. During a **POWER** OFF state the motors maybe moved and position lost. If equipped with encoders they are still powered and monitored.

Reply: If there are no errors, the positive reply "***:A***" will be sent followed by the current state.

Example: **POWER OFF**

:A OFF<eol>

In this example, the power is turned off. All power to the motors are removed and displays turned off.

See also: **MOTCTRL** to turn off just one axis.

Command: RDSBYTE

Shortcut: **RB**

Format: **RDSBYTE** *axis* [*axis*] [*axis*]

Function: Requests the axis Status Byte. The number is one byte, which can be broken down into 8 bits that represent the following internal flags:

Bit 0: 0 = No commanded move is in progress. 1 = A commanded move is in progress. This bit is synonymous with the **STATUS** command. If the bit is set, then **STATUS** returns '**B**', otherwise **STATUS** returns '**N**'.

Bit 1: 0=servo is OFF. 1=servo is ON.

Bit 2: 0=motor phases are turned OFF. 1=motor phases are turned ON.

Bit 3: 0 = Joystick/Knob disabled, 1 = Joystick/Knob enabled

Bit 4: 0 = Motor not ramping, 1 = Motor ramping

Bit 5: 0 = Ramping up, 1= Ramping down

Bit 6: Upper limit switch: 0 = Not Active, 1 = Active

Bit 7: Lower limit switch: 0 = Not Active, 1 = Active

Reply: *<byte as hexadecimal>*

Examples: **RB X**

:<0x8A>

RB X Y

:<0x8A><0x02>

The X axis value 0x8A means the following:

B7: 1 - Axis is at lower limit

B6: 0 - Upper limit switch not active

B5: 0 - Ramping down if ramping

B4: 0 - Not ramping

B3: 1 - Joystick is enabled

B2: 0 - Motor power is off.

B1: 1 - Servo is on

B0: 0 - No commanded move is in progress

See also: **RDSTAT**

Command: RDSTAT

Shortcut: **RS**
Format: **RDSTAT** *axis* [*axis*] [*axis*]
Function: Same as **RDSBYTE**, except the data is returned in ASCII integer format.

Examples: **RS X**
:A 138<eol>

Command: RDSTAT2

Shortcut: **RS2**
Format: **RS2** *axis* [*axis*] [*axis*]
Function: Second form of byte status. Addition Axis bits. current state bits not saved in eprom.

Bit 0: 1=> Hardware present, 0= no axis hardware
Bit 1: 1=> Power on, 0=Power Off
Bit 2: 1=> Homed, 0=not homed since last reset (no softlimits, no lincomp)
Bit 3: 1=> spin on, 0=spin off or 0
Bit 4: 1=> unidir active bit, 0=> not currently active
Bit 5: ELatchSendBit Auto send Index
Bit 6: ELatchActBit Send Encoder Latch Bit 1 => need to send latch data
Bit 7: AutoWhereBit, Auto send Where

Examples: **RS2 Y**
:A 3<eol>

B0: 1=> Hardware present
B1: 1=> Power on
B2: 0=>not homed since last reset (no softlimits, no lincomp)
B3: 0=spin off or 0
B4: 0=> not currently active
B5: 0=>No ELatchSendBit Auto send Index
B6: 0=>No ELatchActBit Send Encoder Latch Bit 1 => need to send latch data
B7: 0=>No auto where send

Command: RESET, REMRES

Shortcut: ALT[255] B or 0xFF 0x52
Format: **RESET**
Format: **REMRES**
Function: This command causes the controller to do a software reset. Upon receiving this command the interface will restart from a power up condition, which will reset the other modules. This command accomplishes the same task as if interface reset button is pressed.
Reply: If there are no errors, a positive reply of "***:A***", followed by the startup sequence.

Example: **RESET<cr>**
:A<eol>

Command: SCANH

Shortcut:

Format: **SCANH** *Rows Columns*

Function: The purpose of this command is to scan the stage in the horizontal direction. Prior to this command, the stage must be moved to the reference point. As defined by:
Ref X = Edge of first column
Ref Y = Center of first row.
The scan starts from the current location and is defined by motion in the X axis
Xmotion = Columns*XPitch
During the Motion, The outbit1 becomes active during the x motion.
At the end of the X Motion the Y axis moves YPitch amount. Outbit2 becomes active during the Y motion. This is repeated for each Row**.**
This command overrides **WINTERVAL** X=XPitch. you must manual enable **WMODE** X+ if you want the XIndex to trigger.

Reply: If there are no errors, a positive reply of "***:A***" is sent back.

Example: **SCANH 10 20<cr>**
:A<eol>
In this example, the stage will start scaniing X travel =20*XPitch (from the **SETSCAN**) and the Y travel = 10 * YPitch. Use **STATUS** to monitor completion.

See also: **SETSCAN, STATUS, WINTERVAL**

Command: SCANV

Shortcut:

Format: **SCANV** *Rows Columns*

Function: The purpose of this command is to scan the stage in the Vertical direction. Prior to this command, the stage must be moved to the reference point. As defined by:
Ref X = Center of first Column.
Ref Y = Edge of first row
The scan starts from the current location and is defined by motion in the Y axis
Ymotion = Rows*YPitch
During the Y Motion, Outbit1 becomes active during the Y motion
At the end of the Y Motion the X axis moves XPitch amount. Outbit2 becomes active during the X motion. This is repeated for each Column**.**
This command overrides **WINTERVAL** Y=YPitch. you must manual enable **WMODE** Y+ if you want the YIndex to trigger.

Reply: If there are no errors, a positive reply of "***:A***" is sent back.

Example: **SCANV 10 20<cr>**
:A<eol>
In this example, the stage will start scaniing Y travel =10*YPitch (from the **SETSCAN**) and the X travel = 20*XPitch. Use **STATUS** to monitor completion.

See also: **SETSCAN, WINTERVAL, WMODE**

Command: SERNUM

Shortcut:
Format: **SERNUM** *[axis] [axis] [axis] [Axis#]*
Function: This command will quary the system for the serial number of a specific axis. If no axis is given, the serial number of the controller is returned. You may also use just an axis number (1=Axis1=?X?, 2=Axis2=?Y?, 3=Axis3=?Z?)
Reply: If there are no errors, the positive reply "***:A***" will be sent followed by the serial number(s).

Example: **SERNUM<cr>**
:A 123<eol>
See also: **MANDATE**

Command: SERVO

Shortcut:
Format: **SERVO** [*axis*±] [*axis*±] [*axis*±]
Default: X- Y- Z-
Persistent: Yes
Function: This command is used for motor drivers with encoders installed. It will turn the servo on or off. With the **SERVO** on, the display and positional information will reflect the current enocder clicks not the number of stepper motor steps. When motor is not running and the motor shaft is moved externally, internal software will try to move the motor back to original position. The activation distance is the minimum encoder clicks, which the motor driver has to be moved externally in order for the internal servo to be activated (see **SERVONULL** to set activation distance). For example, if **SERVONULL** is programmed to be 10, then the motor should be turned at least 10 encoder clicks before the servo system activates.
Reply: If there are no errors, a positive reply of "***:A***" is sent back.
Example: **SERVO X+ Y+ Z-**
:A X+ Y+ Z-
In the example, the X and Y axis are set to servo, the Z axis servo is turned off
See also: **SERVOGAIN, SERVONULL, SERVOSTEP**

Command: SERVOGAIN

Shortcut:
Format: **SERVOGAIN** [*[axis=]ServoGain*] [*[axis=]ServoGain*] [*[axis=]ServoGain*]
Default: X=4 Y=4 Z=0
Persistent: Yes
Function: This command is used for motor drivers with encoders installed. The **SERVOGAIN** value is really inversly related to the gain of the servo system. The gain of the servo system is Gain / (2 ^ SERVOGAIN). The range of values is 0 to 8. Use this value to affectively turn down the Gain for a perticular axis of the servo system.
Reply: If there are no errors, a positive reply of "***:A***" is sent back with the current settings.

Example: **SERVOGAIN Y=3**
:A 0 3 0
In the example, the Y servogain is set to 3 => (Gain for the Y axis is / 2^3 or Gain / 8). XZ are each 0 => Gain / (2^0) = Gain / 1 = Gain
See also: **SERVO, SERVOSTEP, SERVONULL**

Command: SERVONULL

Shortcut:
Format: **SERVONULL** [*axis=ActivationDistance*] [*axis=ActivationDistance*] [*axis=ActivationDistance*]
Default: X=5 Y=5 Z=0
Persistent: Yes
Function: This command is used for motor drivers with encoders installed. It will set the activation distance for the servo system, the range for this value is 0 to 255. This command works in conjuction with **SERVO**. When motor is not running and the motor shaft is moved externally, internal software will try to move the motor back to original position. The activation distance is the minimum encoder clicks, which the motor driver has to be moved externally in order for the internal servo to be activated (see **SERVO** to turn ON/OFF the servo system). For example, if **SERVONULL** distance is programmed to be 10, then the motor should be turned at least 10 encoder clicks before the servo system activates.
Reply: If there are no errors, a positive reply of "***:A***" is sent back with the current settings.

Example: **SERVONULL X=10**
:A 10 2 2
In the example, the X activation distance is set to 10, the Y and Z distances were previously set to 2
See also: **SERVO, SERVOGAIN, SERVOSTEP**

Command: SERVOSTEP

Shortcut:
Format: **SERVOSTEP** [*axis=Steps*] [*axis=Steps*] [*axis=Steps*]
Default: X=10 Y=10 Z=80
Persistent: Yes
Function: This command is used for motor drivers with encoders installed. The servo subsystem requires this value to be set to the number of microsteps per encoder click. The Value must be in the range of 1 to 255. There are 40,000 microsteps per revolution of the stepper motor. This value must be accurate to <+-1 step per encoder click. To calculate this value take the distance traveled in one rotation of the stepper motor. Determine the number of encoder click per that distance and divide 40000 / encoder clicks. The servo system uses this value to estimate the number of microsteps (+-1) to take per pulse to get to the next encoder click.
Reply: If there are no errors, a positive reply of "***:A***" is sent back with the current settings.

Example: **SERVOSTEP X=5**
:A 5 10 10
In the example, the X servo step size is 5 => (5 microsteps=1 encoder click). YZ are set to 10 each.
See also: **SERVO, SERVOGAIN, SERVONULL**

Command: SERVOTIMEOUT

Shortcut:
Format: **SERVOTIMEOUT** [*msec*]
Default: 1000 (timeout after 100 milliseconds)
Persistent: Yes
Function: This command is used for motor drivers with encoders installed. This method works in conjuction to the SERVO command. It defines the end of move command behavior. It is the time the system tries to get into the null position before it gives up. In systems with a large amount of oscillations the control may never get to the null positions.

Reply: If there are no errors, a positive reply of "***:A***" is sent back with the current settings.
Example: **SERVOTIMEOUT** 1000
:A 1000
In the example, the SERVOTIMEOUT command is set to 1000msec
See also: **SERVO, STATUS**

Command: SETSCAN

Shortcut:
Format: **SETSCAN** [*XPitch* [*YPitch*]]
Default: 5.0 5.0
Persistent: Yes
Function: This command sets up the parameters for the ScanH and ScanV commands.Please refer to each of these commands to determine how the parameters are used.
Reply: If there are no errors, the positive reply "***:A***" will be sent followed by the current state.
Example: **SETSCAN 1.0 2.5<cr>**
:A 1.0 2.5<eol>
In the example, the XPitch=1.0 YPitch=2.5 in current units set by **COMUNITS**
See also **SCANH, SCANV, COMUNITS**

Command: SPEED

Shortcut: **S**
Format: **SPEED** *axis=speed* [*axis=speed*] [*axis=speed*]
Default: Depends on axis type (usually set from from last axis eeprom or defaults command)
Persistent: No (current setting is not persistent, but last axis or last default is)
Function: Sets the maximum speed at which the stage will move. Speed is set in COMUNITS per second. Internally this value is set to pulses per second. Minimum speed is 1 pulse / second. Maximum speed is = 400,000 pulses/sec. The actual maximum speed is usually less then this and dictated by the mechanical system. It should be noted that maximum value depends on the external factors like load, motor type etc. It should be programmed with a speed that will not cause motor stalling. For Ballscrew stages a setting of 24.0 mm/sec might be reasonable, but for a rack and pinion stage a setting of 50 mm/sec would be more appropriate.
Reply: If there are no errors, a positive reply of "***:A***" is sent back with current settings.
Example: **S X=24.0**
:A 24.0 24.0 .24
In the example, the maximum speed is set to 24.0 mm/sec 24.0 mm/sec .24mm/sec (assuming COMUNITS in mm)

Command: SPIN

Shortcut: @

Format: **SPIN** *axis=rate* [*axis=rate*] [*axis=rate*]

Default: Power up default is 0, 0, 0 (all axis off)

Function: Spin motor with the speed specified in units (see COMUNITS) per second, range is 1 to 400,000 pulses per second in the low level hardware. This means the smallest SPIN number is the minimum stepper increament per second. This command is reloadable while motor is already rotating. Speed is a signed integer number. If speed is positive, motor is rotated toward ascending numbers, and if it is negative, motor is rotated toward descending numbers. The motor will spin without stopping. The direction can be changed without first stopping motors. In such case motor is ramp down to starting speed and then reversed direction up to programmed speed with internal firmware.

Reply: If there are no errors, a positive reply is sent back. This reply does not signal the end of rotation. The status of motor can be determined by reading the status byte (see Motor Status Byte Read).

Example: **SPIN X=10.0 Y=-.001 Z**

:A

In this example, (assuming COMUNITS is MM) the X-axis will turn at a motor rate of 10 mm per second in the positive direction, the Y-axis at the rate of 1 micron per second in the negative direction, and stop any spin rotation of the Z-axis.

NOTE: To stop rotation, give a value of zero, or just the type the axis letter without an assignment as shown in the example above, or use the **HALT** (\ ) command to stop all axes.

NOTE: The **HALT** command will not return an ***:N-21*** when stopping a **SPIN** command.

Command: SOFTLIMITS

Shortcut:

Format: **SOFTLIMITS** [*axis*±] [*axis*±] [*axis*±]

Default: X- Y- Z-

Persistent: Yes

Function: This command enables / actives softlimits. Softlimits are software limits that behave just like the limits switches. SOFTLIMITS are OR'd with the physical switch limits. Softlimits can be enabled or disabled, but are not active until a home for the axis is performed. (see HOME). once an axs is homed the SOFTLIMITS can be activated or deactivated. Use STROKEMINUS and STROKEPLUS to set the limits of travel.

Reply: If there are no errors, a positive reply of "***:A***" will be followed by the current state

Example: **SOFTLIMITS X+<cr>**

:A X+ Y- Z-<eol>

The command in this example will change the X axis softlimits to enabled, when a home is performed they will become active.

See also: **STROKEMINUS, STROKEPLUS, HOME**

Command: SOLENOID

Shortcut:
Format: **SOLENOID** (Requires additional Solenoid hardware)
Function: This command is for controllers equipped with a solenoid driver. The function actives the solenoid for a period of time (see NOIDDELAY)
Reply: If there are no errors, a positive reply of "***:A***" is sent back.

Example: **SOLENOID<cr>**
:A<eol>
In this example the solenoid is activated.
See also: **NOIDDELAY**

Command: STATUS

Shortcut: **/**
Format: **STATUS**
Function: Inquires regarding the motor status of all axes. Queries the controller whether or not any of the motors are still busy moving following a serial command. (Move, Home, etc...)
Reply: The positive reply can come in two forms:
N - there are no motors running from a serial command
B - BUSY there is a motor running from a serial command
Example: **MOVE X=12345**
:A
STATUS
B
/
B
/
N
In this example, the command **MOVE** started the X-axis moving towards the position 12345. The first and second **STATUS** command returned a "***B***" showing that the motor is still busy moving towards the target. The third time, the **STATUS** command returned an "***N***" signifying that the **MOVE** command is finished and there is no longer any motor movement.

Command: STEPSIZE

Shortcut:

Format: **STEPSIZE** [*axis=stepsize*] [*axis=stepsize*] [*axis=stepsize*]

Default: X=2 Y=2 Z=2

Persistent: Yes

Function: This command changes the stepper motor step size in multiples of 1/40000 steps per rotation. The stepper motors have a maximum of 40,000 microsteps per rotation. The STEPSIZE change the size of one step in multiple units of 40000 / rev. The Number of microsteps per revolution is 40000 / STEPSIZE. For example, a STEPSIZE setting of 2 means there are 20,000 steps per rotation. The range of values for STEPSIZE is 1 to 255. A setting of 2 with a 2mm pitch ballscrew will mean a stepsize of 2mm / 20000 = .0001mm = .1um.

Reply: If there are no errors, a positive reply of "***:A***" will be followed by the current state

Example: **STEPSIZE X=10<cr>**

:A 10 2 2<eol>

The command in this example will change the X **STEPSIZE** such that the X axis will have 4000 steps per rotation.

Command: STSPEED

Shortcut:

Format: **STSPEED** *axis=stspeed* [*axis=stspeed*] [*axis=stspeed*]

Default: Depends on axis type (usually set from from last axis eeprom or defaults command)

Persistent: No (current setting is not persistent, but last axis or last default is)

Function: Sets the start/minimum speed at which the stage will move. This command is similar to **SPEED** command with the exception of being the starting speed of acceleration. When motors are moved they will start moving with programmed **STSPEED**, accelerate until programmed top **SPEED** is reached. **STSPEED<=SPEED**. if you attemp to set **STSPEED** above **SPEED** it will be trucated to speed. If **SPEED** is set to a smaller value than **STSPEED**, then **STSPEED** will also be lowered. **STSPEED** is set in COMUNITS per second. Minimum speed is 1 pulse / second. Maximum speed is = 400,000 pulses/sec.

Reply: If there are no errors, a positive reply of "***:A***" is sent back.

Example: **STSPEED X=.05**

:A 0.05 0.05 .0005

In the example, the starting speed of a move command is set to X=0.05 y=0.05 z=.0005 The speed ramps from **STSPEED->SPEED** then **SPEED->STSPEED**

Command: STROKEMINUS

Shortcut:

Format: **STROKEMINUS** [*axis= distance*] [*axis= distance*] [*axis=distance*]

Default: X=0 Y=0 Z=0

Persistent: Yes

Function: This command sets (or retrieves) the amount of stroke in the negative direction in **COMUNITS.** This is the maxmimum distance to travel under software limit control in the minus direction. Software limits require the system to be HOMEd once before they are active. These are the miniums that the machine coordinate system can obtain before activating the software limits. Note: the new settings are saved, but will not go into effect until the next home performed.

Example: **STROKEMINUS X=100.0 Y=100.0 Z=10.0 <cr>**
:A 0.0 -1.0 0.0<eol>
If **COMUNITS** in MM, the command in this example will make the controller have a maximum negative stoke of 0.0 for the X-axis, -1.0 stroke for the Y-axis, and 0.0mm for the Z-axis.

See also: **STROKEPLUS, HOME, SOFTLIMITS**

Command: STROKEPLUS

Shortcut:

Format: **STROKEPLUS** [*axis= distance*] [*axis= distance*] [*axis=distance*]

Default: depends on stage type

Persistent: Yes

Function: This command sets (or retrieves) the amount of stroke in the positive direction in **COMUNITS.** This is the maxmimum distance to travel under software limit control in the plus direction. Software limits require the system to be HOMEd once before they are active. These are the maximum that the machine coordinate system can obtain before activating the software limits. Note: the new settings are saved, but will not go into effect until the next home performed.

Example: **STROKEPLUS X=100.0 Y=100.0 Z=10.0 <cr>**
:A 100.0 100.0 10.0<eol>
If **COMUNITS** in MM, the command in this example will make the controller have a maximum positive stoke of 100.0 for the X-axis, 100.0 stroke for the Y-axis, and 10.0mm for the Z-axis

See also: **STROKEMINUS, HOME, SOFTLIMITS**

Command: UNIDIR

Shortcut:
Format: **UNIDIR** [*axis= distance*] [*axis= distance*] [*axis=distance*]
Default: X=0.25 Y=0.25 Z=0
Persistent: Yes
Function: This command sets (or displays) the amount of distance in **COMUNITS** and Direction to overtravle to absorb the backlash in the axis' gearing. This unidir value works with an unidir routine built into the controller. The routine ensures that the controller always approaches the final target from the same direction. A value of zero (0) disables the unidir algorithm for that axis. A positive value means the position will always be approached from the positive side. A negative number means always approach the position for the negative side. The unidir value should be just large enough to over come all backlash. The result of a unidir approach is a very high level of repeatability.
Example: **UNIDIR X=.05 Y=.05 Z=0 <cr>**
:A<eol>
If **COMUNITS** in MM, the command in this example will make the controller move the X and Y axes to a location +50 microns away from the final target before moving to the final target, while the unidir algorithm for the Z axis is disabled.

Command: UNITS (display units)

Shortcut: **UN**
Format: **UNITS** [*MM*][*UM*][*UM1*][*UM01*][*NM*][*INCH*] or [axis=[*MM*][*UM*][*UM1*][*UM01*][*NM*][*INCH*]]
Default: MM
Persistent: Yes
Function: This command will Get \ Set the current units that are displayed on the controller. The units can be set to
MM = millimeters 1mm = 1.000 (display)
UM = Microns 1um = 1. (display)
UM1 = Microns Tenths 1um = 1.0 (display)
UM01 = Microns Hundreths 1um = 1.00 (display)
NM = Nanometers. 1nm =1. (display)
INCH = inches, 1" = 1.0000 (display)
Each axis can have a different setting. Note: if an axis is a rotary axis it will always respond with DEG.
Reply: If there are no errors a positive response is sent back with the current setting.
Example: **UNITS MM<cr>** The display units are changed to millimeters (all axis).
:A MM MM MM<eol>
Example: **UNITS Z=UM1<cr>** The display units are changed to .1 micron for the z axis only
:A MM MM UM1<eol>

See also: **COMUNITS, ROTAXIS**

Command: VERSION

Shortcut: **V**
Format: **VERSION**
Function: Requests controller to report which firmware version it is currently using.
Reply: If there are no errors, a positive reply of "***:A***" will be returned, followed by the version number.
Example: **V**
:A Version: H J 4.0

Command: VMOVE

Shortcut:
Format: **VMOVE** axis= *position* [axis= *position*] [axis= *position*]
Function: This command is identical to MOVE as all move motion is coordinated such that the XYZ axis always end exactly at the same time. (see **MOVE**)
Reply: A positive reply of "***:A***" is sent back when the command is received correctly. Reception of the reply does not mean the end of execution, and the command **STATUS** can be used to determine if the move has been completed.

See also: **MOVE, STATUS**

Command: WHERE

Shortcut: **W**
Format: **WHERE** [*axis*] [*axis*] [*axis*]
Function: Returns the current position of the motor for the axis specified. (if blank all axis' are returned)
Reply: If there are no errors, a positive reply of "***:A***" will be followed by the current position, in current **COMUNITS** (MM, UM, UM1, UM01, NM, INCH)

For this example assume: X=1.234567mm Y=7.654321mm Z=0mm from the origin.
Example: **W X Y Z**

:A 1.234567 7.654321 0.0	(with COMUNITS=MM and DECIMAL ON)
:A 1 8 0	(with COMUNITS=MM and DECIMAL OFF)
:A 1234.567 7654.321 0.0	(with COMUNITS=UM and DECIMAL ON)
:A 1235 7654 0	(with COMUNITS=UM and DECIMAL OFF)
:A 12345.67 76543.21 0.0	(with COMUNITS=UM1 and DECIMAL ON)
:A 12346 76543 0	(with COMUNITS=UM1 and DECIMAL OFF, Ludl compatible)
:A 123456.7 765432.1 0.0	(with COMUNITS=UM01 and DECIMAL ON)
:A 123457 765432 0	(with COMUNITS=UM01 and DECIMAL OFF)
:A 1234567 7654321 0	(with COMUNITS=NM) NM => No Decimal
:A 0.0486 0.3014 0	(with COMUNITS=INCH and DECIMAL ON)
:A 0 0 0	(with COMUNITS=INCH and DECIMAL OFF)

See also: **COMUNITS, DECIMAL**
Note: the order of the X, Y, and Z's specified in the **WHERE** command, is the order in which they are returned. The reporting precision of the **WHERE** command can be changed with the **COMUNITS** command and the Setup Control Commands (below). If fractional decimals cannot be handled by the user's software, use the appropriate **COMUNITS** (UM1=> 1unit =100nanometer) and set **DECIMAL**=OFF.

Command: WHO

Shortcut **N**
Format: **WHO**
Function: Inquires the controller to reply with its name. Allows computer software to automatically determine what instrument is attached at the end of the serial line.
Reply: If there are no errors, the controller will reply with a positive response of "***:A***", followed by its name.

Example: **WHO<cr>**
:A XYZ Stage Controller<eol>

Command: WINTERVAL

Shortcut:
Format: **WINTERVAL** [*axis=interval*] [*axis=interval*] [*axis=interval*]
Persistent: Yes
Function: This command sets the internal 'Window Mode' interval or period. This is the distance the axis must travel before outputing a pulse on the (XYZ)Index pin. No output of the (XYZ)Index pin will occur until the **WMODE** command is issued with X+. This value must be larger than **WSIZE**.
Reply: If there are no errors, a positive response of "***:A***" will be returned with the current state.

Example **WINTERVAL X=1.0<cr>**
:A 1.0 0.0 0.0<eol>
In this example the **WINTERVAL** for the X axis is set to 1.0 (see **COMUNITS** for units).

See also: **WMODE, WSIZE**

Command: WMODE

Shortcut:
Format: **WMODE** [*axis*±] [*axis*±] [*axis*±]
Default: X- Y- Z-
Persistent: No
Function: This turns on or off the 'Window Mode' of each axis control drive circuit. In this mode, an active low pulse is output on the (XYZ)Index pin whenever the distance of **WINTERVAL** is traveled. The length of the output pulse is **WSIZE** wide. If you activate this command with just the axis name the internal counters are reset. The counter reset also occurs at the end of every commanded move. Becareful, The (XYZ)Index pins are used for input and output. Make sure you are not driving the pin as an input while this command is trying to drive it as an output. You could damage the electronic circuit. The moment this command is activated (X+, Y+, or Z+) the output pin is being driven even in the inactive state high state.
Reply: If there are no errors, a positive response of "***:A***" will be returned with the current state.

Example **WMODE X+<cr>**
:A X+ Y- Z-<eol>
In this example the **WMODE** for the X axis is activated. All 'Window Mode' counters are reset and after **WINTERVAL** of motion the XIndex Pin will go active for **WSIZE** distance.

See also: **WINTERVAL, WSIZE**

Command: WSIZE

Shortcut:
Format: **WSIZE** [*axis=size*] [*axis=size*] [*axis=size*]
Persistent: Yes
Function: This command sets the internal 'Window Mode' pulse size or width. This is the distance the (XYZ)Index pin will be active. If this value is 0, the minum pulse width is used (one microstep). This value must be less then **WINTERVAL**. No output of the (XYZ)Index pin will occur until the **WMODE** command is issued with X+.
Reply: If there are no errors, a positive response of "***:A***" will be returned with the current state.

Example **WSIZE X=.5<cr>**
:A 0.5 0.0 0.0<eol>
In this example the **WSIZE** will be active for .5 units (see **COMUNITS** for units) after it is activated using **WMODE** and **WINTERVAL**
See also: **WMODE, WINTERVAL**

Command: ZERO

Shortcut: **Z**
Format: **ZERO** [*axis*] [*axis*] [*axis*]
Function: Writes a zero to the position buffer of one or all axes. Allows the user to set current position as the origin.
Reply: If there are no errors, a positive response of "***:A***" will be returned.

Example **ZERO<cr>**
:A<eol>
After the reply, the display will read zero for each axis.

Example **ZERO Z<cr>**
:A<eol>
After the reply, the display for Z will read zero.

Commands Quick Reference

Command	Shortcut	Description
ACCEL	AC	Get/Set the acceleration.
AUTOCONFIG		ON/OFF Auto configuration of new axis attached to control
AUTOWHERE		Get/Set automatic position transfer state.
AXISINV		Get/Set axis inversions
AXISNAME		Get/Set Axis names (single character only)
AXISMEM		To view the Axis EEPROM
AXISREAD		To retrieve the block of axis EEPROM
AXISWRITE		To write the block of axis EEPROM
BACKLASH	B	Get/Set axis backlash correction motion constant.
BUTTONS		Get/Set Buttons enable (non axis control buttons)
COMUNITS		Get/Set the communication units
DECIMAL		Get/Set decimal mode (Ludl compatible => Decimal OFF)
DEFAULTS		Revert back to manufacture defaults
EINDEX		Get/Set Encoder Index mode
EINDEXPOS		Get/Set Encoder Index Position
ENCODER		Get/Set Encoders mode per axis
EHOLD		Get/Set Encoder hold mode per axis
ELATCH	EL	Get the encoder latch buffer
ELATCHSEND		Get/Set ELatch automatic transfer(Send) mode
EOL		Get/Set End Of Line EOL character(s)
EOM		Get/Set Auto Send End Of Move (EOM) byte
EPROM		Get/Set EPROM data
EPROMREAD		To retrieve a block of controller EEPROM
EPROMWRITE		To write a block of controller EEPROM
EWHERE		Get Encoder position (like Where but for encoders)
HALT	\	Halts all serial commands being executed.
HERE	H	Writes a position to an axis position buffer.
HOME	!	Home one or more axis
HOMEDIS		Home Distance. Amount to retract and reapproach
INBIT(123)		Get the state of the input bits
JERK		Get/Set the jerk amount (derivative of ACCEL)
JOYSTICK	J	Enables/Disables manual control input for an axis.
JSINV		Get/Set Joystick inversions
JSPOS		Get the joystick deflection amount (+-100)
JSROTSCALE		Get/Set Z-rotary encoder scale
JSSPD	JS	Get/Set Max speed for joystick.
LIMITS		Get the current state of the limit sensors
LOWI		Set low Current(I)
MANDATE		Get Manufacture Date
MOTCTRL	MC	Enables/Disables motor control for axis.
MOVE	M	Move to an absolute position.
MOVREL	R	Move a relative amount

Commands Quick Reference

Command	Shortcut	Description
NMPERREV		Get/Set the number of nanometers per revolution of the motor
NOIDDELAY		Get/Set the solenoid on time
OUTBIT		Get/Set output bit states
POWER		Get/Set a low power state
RDSBYTE	RB	Returns a Status Information byte for an axis.
RDSTAT	RS	Same as RDSBYTE, in decimal ASCII format.
RESET	0xFF 0x52	Remote Reset controller
REMRES	0xFF 0x52	Remote Reset controller
SCANH		Scan in the horizontal 'X' direction
SCANV		Scan in the vertical 'Y' direction
SERNUM		Get the controller serial number
SERVO		Turn servo on/off per axis
SERVOGAIN		Set a lower servo gain
SERVONULL		Set the amount of motion to reactivate servo
SERVOSTEP		To set the number of microsteps per encoder click
SETSCAN		To setup for scanning
SPEED	S	Get/Set Maximum Speed / velocity
SPIN	@	Causes axis to SPIN at a given speed
SOLENOID		To activate the solenoid drive
STATUS	/	Returns B-Busy, N-Not Busy.
STEPSIZE		To set the number of 40,000 microsteps per step
STSPEED		Get/Set the startspeed
UNIDIR		Get/Set uni-direction approach values
UNITS	UN	Get/Set display units
VERSION	V	Get the version number of the controller
VMOVE		To move in a vector (same as move on this controller)
WHERE	W	Get current position
WHO	N	To get a name string form this controller
WINTERVAL		Get/Set windowing, sync interval
WMODE		Get/Set windowing mode (on/off per axis)
WSIZE		Get/Set Windowing sync pulse width
ZERO	Z	Set all axes to ZERO / set origin

CONTROL SETUP COMMANDS

Currently, the only way to toggle between the High-Level and the Low-Level command format is through the Setup Control Commands.

The following are special commands used to setup different properties of the controller. The controller recognizes these two-byte commands by their prefix byte 255. These commands mimic the Ludl Interface Control Commands and expand upon them. For compatibility, only **COMUNITS** UM1 (tenth micron) and UM01 (hundreth micron) are provided in Low-Level commands.
Command Description

255 65 Switch to High-Level Command Format
Alt[255] A (note: the post-byte "**A**" must be in upper-case)
255 66 Switch to Low-Level Command Format (this will force the system into **COMUNITS**=UM1, if not in UM1 or UM01 already)
Alt[255] B
255 82 Reset Controller
Alt[255] R
255 72 Return hundredth of a micron precision for High Level **WHERE** command.
Alt[255] H
255 84 Return tenth of a micron precision for High Level **WHERE** command.
Alt[255] T
255 80 Power Down (turn power off to motors and blank display)
Alt[255] P
255 125 Halt all motion
Alt[255] }

LOW LEVEL FORMAT

This serial RS-232 interface is used to hook up the controller to a PC with a protocol that imitates the Ludl Low Level command set. The purpose of the low level protocol is to provide a simple interface between a PC program and the controller, without ASCII conversion. The high level protocol is designed to allow direct human interface capability by displaying all numbers and commands in ASCII characters. The high level format is slower due to the extended transmission of ASCII characters as well as the time consumed converting back and forth from 3 byte memory stored numbers and multiple byte ASCII character numbers stored in strings. The low level format deals strictly with numbers that identify modules, commands, data_size, and data represented in 1 to 6 bytes in 2's compliment form.

Commands are generally broken into five groups. In three special cases the number of data bytes group is omitted to speed up the communication process. Data values are broken into 8-bit bytes for the data length times, and then each byte is sent out through serial channel to the interface, from LSB to MSB. The ASCII colon (:) character is defined as the end-of-command code, and used to terminate the command loading sequence at which time the controller clears the serial buffer and attempts to process the command. If the command has errors and cannot be processed and executed, it is ignored.

The low level format is formed by the following 8 bit bytes:

BYTE1: Axis Identification
BYTE2: Command
BYTE3: Number of data bytes to be exchanged for this command
BYTES 4 thru 9: Data Bytes, mostly in 2's compliment form in the order of: Least Significant Byte, Middle Byte, Most Significant Byte
LAST BYTE: The ASCII colon character '**:**' flags the end of the serial command

All values specified through this section of the manual use the following format:

000000	*Decimal*
0x0000	*Hexadecimal*
Ctrl<A>	*ASCII character pressed with Ctrl held down*
Alt[0000]	*Decimal number typed with Alt held down*
'A' ASCII	*character typed in*

Group 1 /Byte 1: Axis Identifier

This one byte character identifies which axis or control function the command is for. If the end-command ':' is received at this point, then the command is aborted and ignored. (Invalid axis)
X Axis: Dec: 24 (Hex: 0x18) Keyboard: Ctrl<X>
Y Axis: Dec: 25 (Hex: 0x19) Keyboard: Ctrl<Y>
Z Axis: Dec: 26 (Hex: 0x1A) Keyboard: Ctrl<Z>

Group 2 / Byte 2: Command Identifier

This is a single byte instruction code. These codes are listed in this manual. If the end-command ':' is received at this point, then the command is aborted and ignored.

Group 3 / Byte 3: Data Size

This is a single byte that gives the number of data bytes for this instruction. This value can also be found in command listing for different commands. Although the range of this variable is from 0 to 255, the controller only supports 0 to 6. Exceptions: There are 3 commands that do not use this data group: '?'-request status, 'G'-start motor / function, 'B'-stop motor / function.

Group 4 / Bytes 4-?: Data Bytes

This group holds the data for the command whether the command is sending or receiving information. The number of bytes for this group varies with each command and is stated in Group 3. Numerical information is broken down into the 8 bit bytes. These are transmitted in the order of Least **S**ignificant **B**yte, **M**iddle **B**yte(s), then **M**ost **S**ignificant **B**yte. Here is how to convert the bytes (up to 4 bytes of numerical data) Conversion:
Value = (Byte1<<0)+(Byte2<<8)+(Byte3<<16)+(Byte4<<24)
Value = (Byte1*(2^0))+(Byte2*(2^8))+(Byte3*(2^16))+(Byte4*(2^24))
Value = Byte1+(Byte2*(256))+(Byte3*(256*256))+(Byte4*(256*256*256))
Value = Byte1+(Byte2*256)+(Byte3*65536)+(Byte4*16777216)

Group 5: Last Byte

This is a one-byte end-of-command character '**:**'. It is a fix predetermined character and set to be decimal (58) value. This byte will end the command. Until this byte is received the interface will not execute the command but will keep the bytes in a buffer. Any information between Group 4 and the ':' is ignored.

COMMAND LISTING

The following are commands formatted by the controller shown in Decimal, and keyboard / ASCII form. The first command, Read Status, give examples that explain in depth the formatting which will be used for the rest of the examples.

Command: Read Status
Dec: 63 **Hex:** 0x3f **Keyboard:** ? **Data Size:** None
Description: The controller will respond to this command in the following manor. If there is a command being executed and the axis motor is enabled, the controller will return an upper case **B**. Otherwise, it will return a lower case **b**.

Example: Command: 24 63 58
Reply: *66*

The above is an example of a stream of bytes that a PC would send serially to the controller and the controller's reply. In the above example the 24 represents the X axis, the 63 represents the Read Status command and the 58 is the colon which signifies the end of the command. The reply 66 is the decimal code for the ASCII character B, which means the axis is currently busy. It should be noted that if a device, which is addressed, does not exist the response to this command will always be the device busy code (66). If the value of the byte received is (98), then device is not busy.

Example: Ctrl<X>?:*b*

The above example shows a way to enter this command using a terminal screen where the Ctrl<X> means that the Ctrl key is held down while the key capital X is pressed. This enters the axis identifier for the X axis. The '?' stands for the command Read Status and the ':' signifies the end of the command. The b is the controller's response, which means the axis is not busy. Notice that with the low level command set there is no spaces, carriage returns or line feeds. Note that, for the sake of easy recognition of the computer response, all computer responses in this manual will be either labeled so, or be printed in *italics*.

Command: Read Motor Position
Dec: 97 **Hex:** 0x61 **Keyboard: a** **Data Size:** 3 or 4
Description: Requests the controller to respond with the current stage position in two's compliment form using 3 bytes. The response is in tenths of microns or hundreths of micron (see 'Control Setup Commands'). The number of bytes requested may be as large as 4 (32 bits).

Example: Command: 24 97 03 58
Reply in Dec: ***160 134 01***

The above is an example of a stream of bytes that a PC would send serially to the controller and the controller's reply. In the above example the 24 represents the X axis, the 97 represents the Read Motor Position command. The 3 means that the controller should return 3 bytes of data, and the 58 is the colon, which signifies the end of the command. In the reply are three bytes: lsb:160, the mb:134, and the msb:01.

Conversion: 160+(134*256)+(1*256*256)=100000 tenths (or Hundreths) of a micron or 10 (1) millimeters from the origin.

The example below shows the same example above as it would appear on a computer serial port terminal program such as Hyperterminal (see command 63 for this manuals formatting information). As can be seen the numbers 160 134 01 correspond to non-legible ASCII characters. For this reason it is difficult to use a terminal program with the low level command set.

<X>a<C>: ***áå_***

Note: As can be seen in the Read Motor Position Command, many low level commands are incompatible with terminal screens, so no terminal screen example will be given throughout the rest of the manual for those commands.

Command: Read Increment Value
Dec: 100 **Hex:** 0x64 **Keyboard:** d **Data Size:** 3
Description: Requests the controller to respond with current setting for the distance of increment moves. The number is a three byte two's compliment number representing a position offset in tenths (hundreths) of a micron.

Example: Command: 24 100 03 58
Reply in Dec: ***160 134 01***

The above is an example of a stream of bytes that a PC would send serially to the controller and the controller's reply. In the above example the 24 represents the X axis, the 100 represents the Read Increment Value command. The 3 means that the controller should return 3 bytes of data, and the 58 is the colon, which signifies the end of the command. In the reply are three bytes: lsb:160, the mb:134, and the msb:01.
Conversion: 160+(134*256)+(1*256*256)=100000 tenths of a micron or 10 millimeters from the origin.

Command: Read Identification
Dec: 105 **Hex:** 0x69 **Keyboard:** i **Data Size:** 6
Description: Requests the controller to respond with the identification code for the Axis Id. The response for X, Y, and Z axis' is EMOT :. The fifth byte is a space (ASCII code 32). The sixth byte is ':' Note: The controller does not support consecutive 105 commands to read the version information.
Example: Command: 24 105 58

Reply in Dec: ***69 77 79 84 32 58***
Reply Converted to ASCII: ***EMOT_:***

The above is an example of a stream of bytes that a PC would send serially to the controller and the controller's reply. The example below shows the same example above as it would appear on a computer serial port terminal program such as Hyperterminal (see command 63 for this manuals formatting information).
<X>i:***EMOT_:*** ('_' = space character)

Command: Read Motor Position and Status
Dec: 108 **Hex:** 0x6C **Keyboard:** l **Data Size:** 4
Description: Requests the controller to respond with the current stage position in two's compliment form using 3 bytes followed by the status byte. This commands force the data member to be 4 bytes, always 3 positional data and one status. The response is in tenths (or hundreths see Controller Setup Commands) of microns. See command 126 (Read Status Byte) for more information on the status byte.

Example: Command: 24 108 04 58
Reply in Dec: ***160 134 01 20***

The above is an example of a stream of bytes that a PC would send serially to the controller and the controller's reply. In the above example the 24 represents the X axis, the 97 represents the Read Motor Position command. The 3 means that the controller should return 3 bytes of data, and the 58 is the colon, which signifies the end of the command. In the reply are three bytes, the lsb:10, the mb:1, and the msb:2, plus the status byte. This can be translated as follows: 160+(134*256)+(1*256*256)=100000 tenths (or hundreths) of a micron or 10 (1.0) millimeters from the origin. See command 126 (Read Status Byte) for more information on the status byte.

Command: Read Start Speed
Dec: 114 **Hex:** 0x72 **Keyboard:** r **Data Size:** 3
Description: Requests the controller to respond with current setting for the speed at which to start a move at the beginning of a ramp up. The number returned is a speed in pulses per second. The range is 1 to 400,000.

Example: Command: 24 114 03 58
Reply in Dec: ***242 03 00***

The above is an example of a stream of bytes that a PC would send serially to the controller and the controller's reply. In the above example the 24 represents the X axis, the 114 represents the Read Start Speed command. The 03 means that the controller should return 3 bytes of data, and the 58 is the colon, which signifies the end of the command. The reply is made up of an lsb, mb, and msb, which would convert as follows:
242+(3*256)+(0*256*256) =1010 pulses per second

Command: Variable Speed Rotating (SPIN)
Dec: 47 **Hex:** 0x2F **Keyboard:** / **Data Size:** 3
Description: This command will rotate the motors without specifying any other target or speed. This number is a 3 byte signed integer, which specifies the direction and the speed of the rotation in pulses per second. The value should be in the range of 0 (stop) to 400000. When this number is positive motor will rotate towards bigger numbers and when it is negative will rotate towards smaller numbers. This command is reloadable. If the motor is already rotating it will ramp up or down depending the last speed loaded. There is no target number to stop the rotation of the motor. The ramp value may also be reloaded to reflect the new ramping time. Motor will stop if one of the following event is occurred:

1. A stop command is received.
2. An end limit switch is closed.
3. The same command is received with the speed equal to zero.

Example: Command: 24 47 03 16 39 0 58
Reply: None

The above is an example of a stream of bytes that a PC would send serially to the controller and the controller's reply. In the above example the 24 represents the X axis, the 47 represents the Variable Speed Rotating (spin) command, the 03 means three bytes of data, and the 58 is the colon, which signifies the end of the command. There is no reply.
16+(39*256)+(0*256*256) =10,000 pulses per second.
In this example the X axis motor will spin toward the positive direction at a speed of 10,000 pulses per second. It should be noted that the maximum speed specified above is the internal limit and does not necessarily means the running speed of the motor installed in the system. It is left to the user to limit the actual maximum running speed.

Command: Read Maximum Speed
Dec: 115 **Hex:** 0x73 **Keyboard:** s **Data Size:** 3
Description: Requests the controller to respond with current setting for the maximum speed the stage is allowed to move. The number returned is a straight two-byte number representing a speed in pulses per second.

Example: Command: 24 115 03 58
Reply in Dec: ***160 134 01***

The above is an example of a stream of bytes that a PC would send serially to the controller and the controller's reply. In the above example the 24 represents the X axis, the 115 represents the Read Maximum Speed command. The 03 means that the controller should return 3 bytes of data, and the 58 is the colon, which signifies the end of the command. The reply is made up of an lsb and msb, which would convert as follows:
160 + (134*256)+ (1*256*256) = 100,000 pulses per second

Command: Read Target Position
Dec: 116 **Hex:** 0x74 **Keyboard:** t **Data Size:** 4
Description: Requests the controller to respond with current target position. The number is a four, three two or one byte, two's compliment, number representing a position offset in tenths (or hundreths, see 'Controller Setup Commands') of a micron.

Example: Command: 24 116 03 58
Reply in Dec: ***160 134 01***

The above is an example of a stream of bytes that a PC would send serially to the controller and the controller's reply. In the above example the 24 represents the X axis, the 116 represents the Read Target Position command. The 3 means that the controller should return 3 bytes of data, and the 58 is the colon, which signifies the end of the command. In the reply are three bytes: lsb:160, the mb:134, and the msb:01.
Conversion: 160+(134*256)+(1*256*256)=100000 tenths of a micron or 10 millimeters from the origin.

Command: Read Joystick Deflection Value
Dec: 122 **Hex:** 0x7A **Keyboard:** z **Data Size:** 1
Description: Requests the controller to respond with current joystick deflection value. This is a signed 2-compliments value. The default position is 0 and maximum deflection is 100. Value range is -100 -> 100.

Example: Command: 24 122 01 58
Reply in Dec: ***100***

The above is an example of a stream of bytes that a PC would send serially to the controller and the controller's reply. In the above example the 24 represents the X axis, the 122 represents the Read Joystick Deflection value. The 01 means that the controller should return 1 byte of data, and the 58 is the colon, which signifies the end of the command. The reply is made up of an 1 byte 100 which means the joystick is at the maximum positive deflection position.

Command: Read Status Byte
Dec: 126 **Hex:** 0x7E **Keyboard:** ~ **Data Size:** 1
Description: Requests the controller to respond with the Status Byte. The number is one byte, which can be broken down into 8 bits that represent the following internal flags:

Bit 0: 0 = No Motor Signal, 1 = Motor Signal (i.e., axis is moving)
Bit 1: 0=servo is OFF. 1=servo is ON.
Bit 2: 0 = Motor phases are turned Off, 1 = Motor phases are turned On
Bit 3: 0 = Joystick/Knob disabled, 1 = Joystick/Knob enabled
Bit 4: 0 = motor not ramping, 1 = motor ramping
Bit 5: 0 = ramping up, 1= ramping down
Bit 7: Upper limit switch: 0 = open, 1 = closed
Bit 6: Lower limit switch: 0 = open, 1 = closed

Example: Command: 24 126 58
Reply:

The above is an example of a stream of bytes that a PC would send serially to the controller and the controller's reply. In the above example the 24 represents the X axis, the 126 represents the Read Status Byte command. The 58 is the colon, which signifies the end of the command. The reply can be broken into its individual bits as follows:

B7: 1 - Axis is at lower limit
B6: 0 - Lower limit switch inactive
B5: 0 - Ramping down if ramping
B4: 0 - Not ramping
B3: 1 - Joystick is enabled
B2: 1 - Motor has power
B1: 1 - Servo Encoders are in use
B0: 0 - Motors are not turned on

Command: Start / Enable Motor (goto target position)
Dec: 71 **Hex:** 47 **Keyboard:** G **Data Size:** 0
Description: Enables the function. Mainly used to turn on / start / enable the motor for an axis specified. This command starts the motor(s) for a move toward the target position. Does not give or receive data so the data field is omitted and the end character ':' follows directly.

Example: Command: 24 71 58
Response: There is no response

Command: Stop / Disable Motor
Dec: 66 **Hex:** 42 **Keyboard:** B **Data Size:** 0
Description: Disables the function. Mainly used to turn off / stop / disable the motor for an axis specified. Does not give or receive data so the data field is omitted and the end character ':' follows directly.

Example: Command: 24 66 58
Response: There is no response

Command: Write Motor Position
Dec: 65 **Hex:** 0x41 **Keyboard:** A **Data Size:** 3
Description: Requests the controller to write the given position to the current position count buffer. The position is given in two's compliment form using 4,3 ,2 or 1 byte(s). The number represents the position in tenths (hundreths see Controller Setup Commands) of microns.

Example: Command: 24 65 03 160 134 01 58
Reply: There is no reply

The above is an example of a stream of bytes that a PC would send serially to the controller. The 24 represents the X axis, the 65 represents the Write Motor Position command. The 3 means that the controller should read three bytes of data. The three bytes are: lsb:160, the mb:134, and the msb:01. The 58 is the colon which signifies the end of the command.

Conversion: 160+(134*256)+(1*256*256)=100000 tenths of a micron or 10 millimeters from the origin.
Reverse Conversion:

10 millimeters*10,000 to get tenths of microns=100,000
lsb = remainder of 100,000 / 256 = 160
mb1 = remainder of 100,000 / 256 / 256 = 134
mb2 = remainder of 100,000 / 256 / 256 / 256 =1
msb = remainder of 100,000 / 256 / 256 / 256 / 256 =0

Or if in hundreths mode:
10 millimeters*100,000 to get hundreths of microns=1,000,000
lsb = remainder of 1,000,000 / 256 = 64
mb1 = remainder of 1,000,000 / 256 / 256 = 66
mb2 = remainder of 1,000,000 / 256 / 256 / 256 =15
msb = remainder of 1,000,000 / 256 / 256 / 256 /256 = 0

Command: Write Target Position (Move Data)
Dec: 84 **Hex:** 0x54 **Keyboard:** T **Data Size:** 4
Description: Requests the controller to write the given position to the target position buffer. The position is given in two's compliment form using 4 bytes. Fewer then 4 bytes are allowed (Data size 3 for example). The high bit in the last byte will be sign extended into the higher bytes. The number represents the position in tenths of microns (or hundreths see Control Setup Commands and COMUNITS).

Example: Command: 24 84 04 160 134 01 00 58
Reply: There is no reply

The above is an example of a stream of bytes that a PC would send serially to the controller. The 24 represents the X axis, the 84 represents the Write Target Position command. The 4 means that the controller should read four bytes of data. The four bytes are: lsb:160, mb1:134, mb2: 01, and the msb:00. The 58 is the colon which signifies the end of the command. Since the MSB is zero, this could have been sent as

Example: Command: 24 84 03 160 134 01 58 (equivalent comand)
Conversion: 160+(134*256)+(1*256*256)+(0*256*256*256)=100000 tenths of a micron or 10 millimeters from the origin.

Reverse Conversion:

10 millimeters*10,000 to get tenths of microns=100,000
lsb = remainder of 100,000 / 256 = 160
mb1 = remainder of 100,000 / 256 / 256 = 134
mb2 = remainder of 100,000 / 256 / 256 / 256 =1
msb = remainder of 100,000 / 256 / 256 / 256 / 256 =0

Or if in hundreths mode:
10 millimeters*100,000 to get hundreths of microns=1,000,000
lsb = remainder of 1,000,000 / 256 = 64
mb1 = remainder of 1,000,000 / 256 / 256 = 66
mb2 = remainder of 1,000,000 / 256 / 256 / 256 =15
msb = remainder of 1,000,000 / 256 / 256 / 256 /256 = 0

Command: Increment Move Up
Dec: 43 **Hex:** 0x2B **Keyboard:** + **Data Size:** 0
Description: Requests the controller to add the Increment Value to the Current Position Value and place the result in the Target Position Buffer. There is no data or response.

Example: Command: 24 43 0 58
Reply in Dec: There is no reply

The above is an example of a stream of bytes that a PC would send serially to the controller. In the above example the 24 represents the X axis, the 43 represents the Increment Move Up command. The 0 means that there is no data. The 58 is the end of command character.

Command: Increment Move Down
Dec: 45 **Hex:** 0x2D **Keyboard:** - **Data Size:** 0
Description: Requests the controller to subtract the Increment Value to the Current Position Value and place the result in the Target Position Buffer. There is no data or response.

Example: Command: 24 45 0 58
Reply in Dec: No reply

The above is an example of a stream of bytes that a PC would send serially to the controller. In the above example the 24 represents the X axis, the 45 represents the Increment Move Down command. The 0 means that there is no data. The 58 is the end of command character.

Command: Write Increment Value
Dec: 68 **Hex:** 0x44 **Keyboard:** D **Data Size:** 4
Description: Requests the controller to write the given position to the Increment Value buffer. The position is given in two's compliment form using 4 bytes. Few then 4 bytes are allowed (Data size 3 for example). The high bit in the last byte will be sign extended into the higher bytes. The number represents the position in tenths of microns (or hundreths see Control Setup Commands and COMUNITS). The Increment Value is used for making successive Relative Moves.

Example: Command: 24 68 03 160 134 01 58
Reply: There is no reply

The above is an example of a stream of bytes that a PC would send serially to the controller. The 24 represents the X axis, the 68 represents the Write Increment Value command. The 3 means that the controller should read three bytes of data. The three bytes are: lsb:160, the mb:134, and the msb:01. The 58 is the colon which signifies the end of the command.

Conversion: 160+(134*256)+(1*256*256)=100000 tenths of a micron or 10 millimeters from the origin.
Reverse Conversion:

10 millimeters*10,000 to get tenths of microns=100,000
lsb = remainder of 100,000 / 256 = 160
mb1 = remainder of 100,000 / 256 / 256 = 134
mb2 = remainder of 100,000 / 256 / 256 / 256 =1
msb = remainder of 100,000 / 256 / 256 / 256 / 256 =0

Or if in hundreths mode:
10 millimeters*100,000 to get hundreths of microns=1,000,000
lsb = remainder of 1,000,000 / 256 = 64
mb1 = remainder of 1,000,000 / 256 / 256 = 66
mb2 = remainder of 1,000,000 / 256 / 256 / 256 =15
msb = remainder of 1,000,000 / 256 / 256 / 256 /256 = 0

Command: Write Start Speed
Dec: 82 **Hex:** 0x 52 **Keyboard:** R **Data Size:** 3
Description: Requests the Controller to write the given speed to the Start Speed buffer. The speed is divided down into two 8-bit bytes by dividing the number down by 256. The number represents the speed in pulses per second. The Start Speed is the speed at which the acceleration ramp starts. The value is an integer between 1 and 400,000. It should be noted that the maximum speed specified is the internal limit and does not necessarily mean the running speed of the motor installed in the system. It is left to the user to limit the actual maximum running speed.

Example: Command: 24 82 3 232 3 0 58
Reply: There is no reply

The above is an example of a stream of bytes that a PC would send serially to the controller. The 24 represents the X axis, the 82 represents the Write Start Speed command. The 3 means that the controller should read three bytes of data. The three bytes are: lsb:232, mb: ,3 and the msb:0. This means the axis will start at 1000 pulses per second. The 58 is the colon which signifies the end of the command.
Conversion: 232+(3*256) + (0*255*256)=1000 pulses per second.

Command: Write Top Speed
Dec: 83 **Hex:** 0x53 **Keyboard:** S **Data Size:** 3
Description: Requests the controller write the given speed to the Top Speed buffer. The speed is divided down into three 8-bit bytes by dividing the number down by 256. The number represents the speed in pulses per second. The Speed is the maximum speed the axis will move. Note: for any given motion this speed may not be obtain if the motion is too short. The value is an integer between 1 and 400,000. It should be noted that the maximum speed specified is the internal limit and does not necessarily mean the running speed of the motor installed in the system. It is left to the user to limit the actual maximum running speed.

Example: Command: 24 83 3 160 134 1 58
Reply: There is no reply

The above is an example of a stream of bytes that a PC would send serially to the controller. The 24 represents the X axis, the 83 represents the Write Top Speed command. The 3 means that the controller should read three bytes of data. The three bytes are: lsb:160, mb: 134, and the msb:1. This means the top speed the axis will travel is at 100000 pulses per second. The 58 is the colon which signifies the end of the command.
Conversion: 160+(134*256)+(1*256*256)=100,000 pulses per second.

Command: Joystick / Control Device Enable
Dec: 74 **Hex:** 0x4A **Keyboard:** J **Data Size:** 0
Description: Enables the control device function. Allows enabling a control device such as a Joystick or Command Knob to be re-enabled.

Example: Command: 24 74 58 **OR** 24 74 0 58
Reply: There is no reply

The above is an example of a stream of bytes that a PC would send serially to the controller. The 24 represents the X axis, the 74 represents the Enable Joystick command. The data size is 0 and can either be included or left off. The 58 is the colon which signifies the end of the command.

Command: Joystick / Controller Disable
Dec: 75 **Hex:** 0x4B **Keyboard:** K **Data Size:** 0
Description: Disables the control device function. Allows disabling a control device such as a Joystick or Command Knob so that no external signals are allowed to affect move functions during PC control.
Example: Command: 24 75 58 **OR** 24 75 0 58
Reply: There is no reply

The above is an example of a stream of bytes that a PC would send serially to the controller. The 24 represents the X axis, the 74 represents the Disable Joystick command. The data size is 0 and can either be included or left off on the controller. The 58 is the colon which signifies the end of the command.

Commands Quick Reference

Command: Axis, CMD, DataLen, Data, ':'

Command: 24 43 0 58	;Increment Move Up
Command: 24 45 0 58	;Increment Move Down
Command: 24 47 3 xx yy zz 58	;Write rotating speed to motor driver (SPIN)
Command: 24 60 58	;Motor On
Command: 24 61 58	;Motor Off
Command: 24 63 58	;Read Status => B=busy, b=not busy
Command: 24 65 03 160 134 01 58	;Write Motor Position
Command: 24 66 58	;Stop / Disable Motor
Command: 24 68 03 160 134 01 58	;Write Increment Value
Command: 24 71 58	;Start / Enable Motor
Command: 24 74 58 OR 24 74 0 58	;Joystick / Control Device Enable
Command: 24 75 58 OR 24 75 0 58	;Joystick / Controller Disable
Command: 24 77 58 OR 24 77 0 58	;Servo On
Command: 24 78 58 OR 24 78 0 58	;Servo Off
Command: 24 81 01 45 58	;Write Ramping Time
Command: 24 82 2 50 0 58	;Write Start Speed
Command: 24 83 2 112 23 58	;Write Top Speed
Command: 24 84 03 160 134 01 58	;Write Target Position (move)
Command: 24 97 03 58	;Read Motor Position => lsb, mb, msb
Command: 24 100 03 58	;Read Increment Value => lsb, mb, msb
Command: 24 105 58	;Read Identification => 'EMOT :' 69 77 79 84 32 32 58
Command: 24 108 03 58	;Read Motor Position and Status => lsb, mb, msb + StatusByte
Command: 24 113 01 58	;Read Ramp Time => byte (Ramp time in ms)
Command: 24 114 02 58	;Read Start Speed => lsb, mb (our speed has 3 bytes)
Command: 24 115 02 58	;Read Maximum Speed => 2 bytes
Command: 24 116 03 58	;Read Target Position => lsb, mb, msb
Command: 24 122 1 58	;Read joystick deflection values (0-100 with sign)
Command: 24 126 58	;Read Status Byte

For warranty repair return the product to the warranty department of Conix Research Inc. at the following location: You should provide a written description of the problem with the unit. Consumer must prepay all postage, shipping, insurance, and delivery costs associated with the return of the product.For more information refer to the Conix Research Inc. Limited Warranty Card provided with this product.

Conix Research, Inc.

857 28th st.
Springfield, OR 97477 USA
Phone: (541) 747-8512
Fax: (541) 747-8528
Web: www.ConixResearch.com
E-mail: Support@ConixResearch.com

Last Revision: January 6, 2006
Version H J 4.0

CONIX RESEARCH Inc
857 28TH SPRINGFIELD, OR 97477
PHONE (541) 747-8512 FAX (541) 747-8528