Peter Norberg Consulting, Inc.
Professional Solutions to Professional Problems
P.O. Box 10987 Ferguson, MO 63135-0987 (314) 521-8808

Information and Instruction Manual for SS0805 and SS0905 Stepper Motor Controllers

By
Peter Norberg Consulting, Inc.
Matches GenStepper Firmware Revision 2.18 and PotStepper Firmware version 3.8

Table Of Contents

Table Of Contents 2
Disclaimer and Revision History 5
Product Safety Warnings 6
LIFE SUPPORT POLICY 6
Introduction and Product Summary 7
Short Feature Summary 8
Firmware Configuration At Time of Ordering Product 9
Default Microstep Size 9
Default Stop Rate 9
Default Ramp Rate 9
Default Auto-Full-Step Rate 9
Default Auto-Full-Step Mode 9
Default Full-Power Level (S1K jumper not installed) 9
Default Low-Power Level (S1K jumper installed) 9
Default Motor Idle Winding Current 10
Default Limit-Switch Stop Mode 10
Default 'E' mode startup (GenStepper only) 10
Default Double-Current Operation 10
Configuring Serial Baud Rate (GenStepper only) 10
Disable Slew Inputs (GenStepper Only) 10
Hardware Configuration 10
Configuring Half-Power Mode (equivalent to the "H" command) 10
Configuring Double Current Mode 11
Board Jumpers 12
Jumper JS – Enable Serial based communications 12
Jumper R1K – Configure 'Double Current' mode 12
Jumper S1K – Configure 'Half power' mode 12
Jumper SS, DS, 5V – Configure Power Supply mode 12
Cooling Requirements 13
Potentiometer Control, when using PotStepper or RelayStepper firmwares 13
Power-On (and reset) Defaults 14
TTL Mode of operation 15
TTL Input Voltage Levels: Schmitt-Triggered or CMOS 15
Input Limit Sensors, lines LY- to LX+ 16
Motor Slew Control: Y- to RDY 17
USB Driver Installation Under Windows for the SS0905 board 18
Base Driver Installation Under Windows 18
Initial testing of the board after driver installation – TestSerialPorts 19

Adjusting Default COM port properties for best operation 20
Serial Operation 21
Serial Commands 22
Serial Command Quick Summary 22
0-9, +, - – Generate a new VALUE as the parameter for all FOLLOWING commands 23
A – Select the Auto-Full Power Step Rate 23
B – Select both motors 23
E – Enable or Disable Remote Direct Pulse Control (GenStepper Only) 24
G – Go to position x on the current motor(s) 26
H – Operate motors at ½ power 27
I – Wait for motor 'Idle' 27
K –Set the "Stop oK" rate 27
L – Latch Report: Report current latches, reset latches to 0 28
M – Mark location, or go to marked location. 29
O – step mOde – How to update the motor windings 29
P – sloPe (number of steps/second that rate may change) 30
R – Set run Rate target speed for selected motor(s) 31
S – start Slew. 32
T – limiT switch control (firmware versions 1.65 and above) 33
V – Verbose mode command synchronization 34
W – Set windings power levels on/off mode for selected motor 35
X – Select motor X 35
Y – Select motor Y 36
Z – Stop current motor. 36
! – RESET – all values cleared, all motors set to "free", redefine microstep. Duplicates Power-On Conditions! 37
= – Define current position for the current motor to be 'x', stop the motor 38
? – Report status 39
other – Ignore, except as "complete value here" 45
More Examples 46
Direct TTL Step Control – GenStepper Firmware Only 47
Basic Stamp™ Sample Code 49
Listing for GENDEMO.BS2 – 9600 Baud, READY line based 50
Listing for GENDEMOSER.BS2 – 9600 baud, serial based 52
Listing for GENSEEKSER.BS2 – 9600 Baud, serial based, complex actions 54
SerTest.exe – Command line control of stepper motors 57
StepperBoard.dll – An ActiveX controller for StepperBoard products 58
Board Connections 59
Board Size 59

Mounting Requirements 59
Connector Signal Pinouts 60
SX-Key debugger connector 60
TTL Limit Input and Reset 61
TTL Motor Direction Slew Control 61
Board status and TTL Serial 62
RS232 Serial (SS0805) 62
USB Serial (SS0905) 63
Power Connector 64
Calculating Current And Voltage Power Supply Requirements 65
1. Determine the individual motor winding current requirements. 65
2. Determine current requirement for actually operating the motor(s) 65
4. Note the logic supply requirements 66
5. Determine the power supplies you will be using 66
Wiring Your Motor 67
Stepping sequence, testing your connection 68
Determining Lead Winding Wire Pairs 69
Sequence Testing 71
Single motor, double current mode of operation 73
Motor Wiring Examples 74
Unipolar Motors 74
Jameco 105873 12 Volt, 0.150 Amp/winding, 3.6 deg/step 74
Jameco 151861 5 Volt, 0.55 Amp/winding, 7.5 deg/step 75
Jameco 155432 12 Volt, 0.4 Amp/winding, 2000 g-cm, 1.8 deg/step 75
Jameco 162026 12 Volt, 0.6 Amp/winding, 6000 g-cm, 1.8 deg/step 75
Jameco 169201 24 Volt, 0.3 Amp/winding, 1.8 deg/step 76
Jameco 173180 12 Volt, 0.060 Amp/winding, 0.09 deg/step geared 76
Jameco 174553 12 Volt, 0.6 Amp/winding, 7.5 deg/step 76

Disclaimer and Revision History

All of our products are constantly undergoing upgrades and enhancements. Therefore, while this manual is accurate to the best of our knowledge as of its date of publication, it cannot be construed as a commitment that future releases will operate identically to this described. Errors may appear in the documentation; we will correct any mistakes as soon as they are discovered, and will post the corrections on the web site in a timely manner. Please refer to the specific manual for the version of the hardware and firmware that you have for the most accurate information for your product.

This manual describes artwork SS0805 and SS0905, both beta and production releases. The firmware releases described are GenStepper version 2.18 and PotStepper version 3.8. Note that if no manual has yet been published which matches a given firmware level, then the update is purely one of internal details; no new features will have been added.

As a short firmware revision history key points, we have:

Version	Date	Description
GenStepper 2.3, PotStepper 3.3	December 14, 2005	Beta release of first SS0805 artwork
GenStepper 2.12	August 9, 2006	SS0805 and SS0905 production artworks, On GenStepper firmware, added option for TTL-control of motor current during step-and-direction mode of operation, as well as order-only option for power-on S1K selection of baud rate, as 'Dual Baud Rate' feature, which replaces option of selection of motor current feature at power on if requested.
GenStepper 2.15 and 2.16	February 1, 2007	Updated to note support for later firmwares.
GenStepper 2.18, PotStepper 3.8	October 31, 2008	Improved serial resynchronization after bad serial data reception

The microstep functionality is generated by a PWM (Pulse-Width-Modified)-like algorithm, and is non-feedback based. Although the software has a demonstrated maximum resolution of $1/64^{th}$ of a full-step, in practice most inexpensive stepping motors will not reliably produce unique positions to this level of precision. Mainly, the microstep feature gives you a very smooth monotonic motor action, with the capability of requesting step rates as slow as $1/64^{th}$ of a full step per second. We strongly suggest use of the default $1/16^{th}$ of a full step microstep size; this seems to give the best performance on most motors that we tested. Most non-microstep enabled stepper motors will experience "uneven" step sizes when microstepped between their normal full step locations; however, the steps are monotonic in the correct direction, and are usually consistently located for a given position value.

Product Safety Warnings

The SS0805 has components that can get hot enough to burn skin if touched, depending on the voltages and currents used in a given application. Care must always be taken when handling the product to avoid touching these components:

- The 2904 5 volt regulator (located on the bottom side of the board, close to a the 4-pin SIP header used to configure power options.
- The ULN2803 18 pin SOIC on the top side of the board near the center.

Always allow adequate time for the board to "cool down" after use, and fully disconnect it from any power supply before handling it.

The board itself must not be placed near any flammable item, as it can generate heat.

Note also that the product is not protected against static electricity. Its components can be damaged simply by touching the board when you have a "static charge" built up on your body. Such damage is not covered under either the satisfaction guarantee or the product warranty. Please be certain to safely "discharge" yourself before handling any of the boards or components.

If you attempt to use the product to drive motors that are higher current or voltage than the rated capacity of the given board, then product failure will result. It is quite possible for motors to spin out of control under some combinations of voltage or current overload. Additionally, many motors can become extremely hot during standard usage – some motors are specified to run at 90 to 100 degrees C as their steady-state temperature.

If you are operating motors that require more than 200 mA of current per winding, then you must provide for fan-based cooling of the board. We suggest at least 8-10 CFM, directed either across the top of the board, or downward towards the board (so that both the 2904 and the ULN2803 are in the direct path of the airflow).

LIFE SUPPORT POLICY

Due to the components used in the products (such as National Semiconductor Corporation, and others), Peter Norberg Consulting, Inc.'s products are not authorized for use in life support devices or systems, or in devices which can cause any form of personal injury if a failure occurred.

Note that National Semiconductor states "Life support devices or systems are devices which (a) are intended for surgical implant within the body, or (b) support or sustain life, and in whose failure to perform when properly used in accordance with instructions or use provided in the labeling, can be reasonably expected to result in a significant injury to the user". For a more detailed set of such policies, please contact National Semiconductor Corporation.

Introduction and Product Summary

*Please review the separate "**First Use**" manual before operating your stepper controller for the first time. That manual guides you through a series of tests that will allow you to get your product operating in the shortest amount of time.*

The **SS0805** unipolar microstepping motor controller from Peter Norberg Consulting, Inc., has the following general performance specifications:

	SS0805	**SS0805**
Unipolar Motor	Yes	Yes
Bipolar Motor	No	No
Maximum Motor supply voltage (Vx and Vy)	26V	26V
Maximum Logic supply voltage (Vc)	15V	15V
Quiescent current (all windings off)	250 mA	250 mA
Maximum winding current (per motor winding, requires external fan to operate)	0.5A	0.5A
Board size	1.0" x 3.8"	1.0" x 3.8"
Dual power supply capable	Yes	Yes
USB	No	Yes
RS232	Yes	No

Each board can be controlled simultaneously via its TTL input lines and its 2400 or 9600 baud serial interface. If the TTL inputs are used alone, then simple pan, tilt, and rate of motion are provided via 5 inputs, configured according to the firmware used (GenStepper or PotStepper); additional lines are used as limit-of-motion inputs. When operated via the serial interface, full access to the controller's extreme range of stepping rates (1 to 62,500 microsteps per second), slope rates (1 to 62,500 microsteps per second per second), and various motor motion rules are provided. Under the GenStepper firmware, a special mode may be enabled which allows an external controller to provide its own step pulses, allowing for up to 62,500 microsteps per second of operation. Under the PotStepper firmware, a user-provided potentiometer may be optionally used to select the rates for the motors. Under both firmwares, the boards have a theoretical microstep resolution of 1/64 of a full step, and use a constant-torque algorithm when operating in microstep mode. Please note that, although 1/64th resolution is theoretically available, most real use should be restricted to 1/16th or 1/8th step due to limitations of the non-current feedback PWM stepping methodology used by the code.

If you are operating motors that require more than 200 mA of current per winding, then you must provide for fan-based cooling of the board. We suggest at least 8-10 CFM, directed either across the top of the board, or downward towards the board (so that both the 2904 and the ULN2803 are in the direct path of the airflow).

The boards themselves have the additional feature of containing provision for in-circuit reprogramming of the Ubicom (Scenix) SX28 chip that is being used as the controller. The Parallax, Inc.^{tm} SX-Key[1] may be used to perform in-circuit reprogramming and debugging of software. ***Note that such action would void the warranty of the product***. This capability is provided as a convenience for those who would like to run different devices (such as three or four phase unipolar steppers) or use different procedures than those for which the product was intended.

[1] Note: SX-Key is a copyrighted product by Parallax, Inc. Please go to their web site at www.parallax.com for more information about this device.

Short Feature Summary

- One or two stepper motors may be independently controlled at one time.
- Each motor must be Unipolar.
- Each motor may draw up to 0.5 amps/winding.
- If only a single motor is connected to the board, then you can configure the board to operate in **DOUBLE POWER** mode. This allows the board to operate a single motor at twice the rated current for the board. For example, the SS0805 0.5 amp product can operate a single 1 amp motor, when this feature is enabled.
- Limit switches may optionally be used to automatically request motion stop of either motor in either direction.
- Rates of 1 to 62,500 microsteps per second are supported.
- Step rates are changed by linearly ramping the rates. The rate of change is independently programmed for each motor, and can be from 1 to 62,500 microsteps per second per second.
- All motor coordinates and rates are always expressed in programmable microunits of up to $1/64^{th}$ step. Changing stepping modes between half, full and micro-steps does not change any other value other than which winding pairs may be driven at the same time, and how the PWM internal software is operated.
- Motor coordinates are maintained as 32 bit signed values, and thus have a range of -2,147,483,647 through +2,147,483,647.
- Both GoTo and Slew actions are fully supported.
- Four modes of stepping the motor are supported:
- Half steps (alternates 1 winding and two windings enabled at a time),
- Full power full steps (2 windings enabled at a time)
- Half power full steps (1 winding enabled at a time)
- Microstep (programmable to as small as $1/64^{th}$ steps, using a near-constant-torque PWM algorithm)
- A TTL "busy" signal is available, which can be used to see if the motors are still moving. This information is also available from the serial connection.
- Simple control of the motors may be done by switch closure. Each motor can be told to slew left or right, or to stop by grounding the relevant input lines. Similarly, the rate of motion can be controlled either via stepping through a standard set of rates via grounding another input (GenStepper firmware) or via a user-provided potentiometer (PotStepper firmware).
- Complete control of the motors, including total monitoring of current conditions, is available through the 2400 or 9600 baud serial connection.
- Under the GenStepper firmware, an additional mode is available which allows an external computer to directly generate step sequences on the motor control lines. Up to 62,500 steps per second may be requested.
- Runs off of a single user-provided 6.5 to 15 volt DC power supply, or two supplies (7.5-15V for the logic circuits and 5-26V for the motors).
- Any number of motors may be run off of one serial line, when used in conjunction with one or more SerRoute controllers.

Firmware Configuration At Time of Ordering Product

Both the GenStepper and PotStepper firmwares have a common set of initial settings that are selected at power-on or reset *that may be preconfigured at the time the product is ordered*. With the exception of the mode of stepping used when the "Auto-full-step" rate is reached, all of these features may be reset through use of the appropriate serial command.

Default Microstep Size

Normally, the firmware defaults to a microstep size of 1/16th of a full step (the equivalent of the "**4!**" command) at power-on or reset. When you order this firmware from us, you have the option of setting this to any of the valid values (1/64, 1/32, 1/16, 1/8, ¼, ½ or full-step).

Default Stop Rate

Normally, the firmware defaults to a stop rate of 80 microsteps per second at power-on or reset (equivalent to the "**80k**" serial command). This can be ordered as any valid stop rate for the system.

Default Ramp Rate

Normally, the firmware defaults to a ramp rate of 8000 microsteps/second/second (equivalent to the "**8000p**" command). This can be ordered as any valid ramp rate for the system.

Default Auto-Full-Step Rate

Normally, the firmware defaults to a rate of 3072 microsteps/second as being the rate at which it selects the "Auto-Full-Step" mode (equivalent to the '**3072A**' command). This can be ordered as any rate which is valid for the system.

Default Auto-Full-Step Mode

Our testing of the product shows that once you exceed a given rate (as defined by the 'Auto-Full-Step Rate' command/setting), you can obtain more torque from the motors by switching to simple full-step operation. By default, the "double winding" mode (equivalent to the "**2o**" command) is selected when this "Auto-Full-Step" rate is reached, as that has worked best with the motors that we have tested. However, the mode used may be defined by you at the time of ordering the product to be any of the modes available from the "o" command. Please see the "A" command for details about the "Auto-Full-Step" mode command.

Default Full-Power Level (S1K jumper not installed)

Normally, we ship the product such that the default code will select full winding current operation (see the "**0H**" command) when the board is reset or powered on and there is no jumper in the S1K position. At the time of ordering the product, you may change this to operate in ½ power mode ("1H") in this case.

Default Low-Power Level (S1K jumper installed)

As with the Full-Power-Level, we also provide an automatic selection of ½ power level (approximately) at the time of board reset (equivalent to the "**1H**" command). This mode may be configured by inserting jumper in the S1K position. You may optionally order this to be the full power level ("0H") if this is better for your application.

Note that for both the high and low power level defaults, the actual current level used can be redefined at any time through use of the "h" command.

Default Motor Idle Winding Current

Normally, at power on or reset, the motor windings are set to be off (no current supplied) whenever motion has completed (equivalent to the "**0W**" command). At the time of ordering the product from us, you may specify the default idle winding mode to be any of our valid values (see the "W" command for details).

Default Limit-Switch Stop Mode

Normally, the firmware defaults to treating a limit-switch input as 'soft'; that is to say, the firmware issues a 'z' command when a limit is reached. This can be ordered as a 'hard' stop – the board will **INSTANTLY** stop the motor when a limit is reached. Note that damage to gear trains is possible if this option is ordered!

Default 'E' mode startup (GenStepper only)

Normally, the firmware defaults starting up with the 'e' command (direct pulse-step-control) disabled. When you order your board, you may request any of the legal 'e' modes to be enabled upon startup.

Default Double-Current Operation

Normally, the firmware is configured to operate two motors independently of each other. The 'Double Current' mode of operation allows one motor to be run at up to twice the rated current of the board, assuming that everything is connected correctly (see the later manual section on double current operation). By default, the 'Double Current' mode is enabled by the R1K jumper option (described elsewhere); however, at the time of ordering, you may request that this mode be the only way that the controller operates. In this case, the jumper option is ignored, and double current mode is permanently enabled.

Configuring Serial Baud Rate (GenStepper only)

As of version 2.0, by default, all serial communications with the GenStepper firmware operate at 9600 baud, 8 data bits, 1 stop bit, no parity. If you need to communicate at 2400 baud, you must order the board from the factory configured with the differing baud rate. Note that earlier versions allowed you to program the baud rate via a jumper option; in version 2.0 that jumper was reassigned.

As of version 2.12, you may special a special option of "DUALBAUD". This option redefines the 1K resistor-to-ground on SO to mean "operate at ½ of the standard baud rate" (instead of 'operate at ½ power'). This allows you to operate the board at either its baud rate as specified as part of your order (by default this would be 9600), or at ½ of that baud rate.

Disable Slew Inputs (GenStepper Only)

As of version 2.5 of the GenStepper firmware, you may order using the "NOSLEW" option. This will disable use of the SLEW inputs as controls of motor slewing, thus providing you with 4 generic TTL inputs.

Hardware Configuration

The GenStepper and PotStepper firmwares have two major features that can be configured as startup options. This means that any combination of these features may be automatically controlled whenever the firmware receives a power-on, hardware reset, or software reset action. The features are selected by adding a shorting plug to on-board jumper locations.

Configuring Half-Power Mode (equivalent to the "H" command)

Half-Power mode allows you to operate motors at higher voltages, while still operating at their nominal current. This can allow you to either operate motors whose nominal voltage is otherwise too low for our products, or to force motors to be able to operate at higher speeds. **Determining the correct voltage to use is a non-trivial task; please see**

the separate manual "Half Power Notes" for full details about this option before attempting to use it!

This mode may be configured by inserting a shorting plug at the S1K location. The hardware selection may be changed at any time through issuing the "1h" or "0h" commands, as described elsewhere in this manual. However, by operating through use of this hardware strap, you are much less likely to ever "blow out" a board by failing to issue the "1h" command after a power-on or reset condition!

Please refer to the "Board Connections" section starting on page 59 for an image which shows the location of the jumpers.

Configuring Double Current Mode

"Double Current" mode allows the controller to operate a single winding motor at up to double the rated level of the board (see the manual section "Single motor, double current mode of operation" for more information about this capability).

You configure the board to operate this way by inserting a shorting plug at the R1K location.

Please refer to the "Board Connections" section starting on page 59 for an image which shows the location of the jumpers.

Board Jumpers

There are four jumpers possible on SS0805 and SS0905 boards. They control serial communications, power, and configure special firmware options.

Jumper JS – Enable Serial based communications

The **JS** jumper is near the center of the board, close to the ULN2803 driver. If installed, then RS232 or USB communications via that connector is enabled. You must NOT use the SI and SO connections when JS is installed, since you will end up with 2 devices driving the same signal (SI), which can eventually destroy one or both devices.

If this jumper is removed, then only TTL-Serial communication will work, via the SI and SO connections.

Jumper R1K – Configure 'Double Current' mode

The **R1K** jumper is located just beside the **S1K** jumper (on the left side of the board). If it is installed, then the board will operate in its special 'double current' mode of operation. Please see the manual section "Single motor, double current mode of operation" for more information about this capability

Jumper S1K – Configure 'Half power' mode

The **S1K** jumper is located just beside the **R1K** jumper (on the left side of the board). If it is installed, then the board will power-on (and reset) to operate in the half-power mode of operation. If it is not installed, then the board will power-on to full power operation.

Jumper SS, DS, 5V – Configure Power Supply mode

This set of jumper options is used to configure the board to accept the power supply voltages that you will be providing. **If you do not provide voltages that match the expectations of the jumper setting, then you may damage the board (and void your warranty)!**

Please refer to the manual section entitled "Power Connector" for information on how to set this jumper.

Cooling Requirements

If you are operating motors that require more than 200 mA of current per winding, then you must provide for fan-based cooling of the board. We suggest at least 8-10 CFM, directed either across the top of the board, or downward towards the board (so that both the 2904 and the ULN2803 are in the direct path of the airflow).

Potentiometer Control, when using PotStepper or RelayStepper firmwares

By default, the firmware is set up to expect the target rate to use for the motors to be controlled by a potentiometer connected to the POT input line and to one of the GND lines.

The firmware uses the time that it takes to discharge a 1uF capacitor (mounted on the board) through the potentiometer to ground as the source of information to use on determining the rate to apply to the motor(s). The rate range is from 1 to XXXX microsteps/second, where XXXX is controlled by the size of the potentiometer.

Resistance (Ohms)	**Approximate Maximum Rate (microsteps/second)**
500	950
1000	2600
2000	6700
5000	16300
10000	30700

The above numbers are approximate, and may be off by as much as 20%. Note that PotStepper attempts to automatically detect whether the potentiometer is present. If the calculated rate would be greater than about 34,000, the code assumes that the potentiometer is missing, and resets the rate to the default rate for the system (usually 800).

By issuing the rate change serial command ("xR", where "x" is the desired rate), each motor may be independently connected or disconnected from the potentiometer. If 'x' is set to 0, then the currently selected motor gets its rate from the current potentiometer position. Otherwise, the requested rate (such as 450 from the command "450r" becomes the new target rate for the given motor.

Power-On (and reset) Defaults

In addition to the above hardware straps, the board acts at power on (or reset) as if the following serial commands have been given:

- **3072A** – Set the Automatic Full Step rate to be >=3072 microsteps/second
- **B** – Select both motors for the following actions
- **0**= – Reset both motors to be at location 0
- **0H** – Set motors to full power mode
- **80K** – Set the "Stop OK" rate to 80 microsteps/second
- **3O** – Set the motor windings Order to "microstep"
- **8000P** – Set the rate of changing the motor speed to 8000 microsteps/second/second
- **800R** – Set the target run rate for the faster motor to 800 microsteps/second. This will be overridden by the potentiometer setting, if one is attached and the PotStepper or RelayStepper firmware is installed.
- **0T** – Enable all limit switch detection
- **1V** – Set <CR><LF> sent at start of new command, no transmission delay time
- **0W** – Full power to motor windings

TTL Mode of operation

The TTL input control method provides for nine input signals and one output signal. TTL based control operates at the same time as serial control; therefore, any of the actions listed below may be requested at any time that the board is not in its special "direct computer control" mode of operation (GenStepper only).

All external connections are done via labeled terminal block connections on the "top" and "bottom" sides of the boards, and one RS232 or USB serial port on the "bottom" of the board.

TTL Input Voltage Levels: Schmitt-Triggered or CMOS

The input voltage levels which are sensed by the TTL input signals to the boards depend on the mode of operation of the board.

All TTL input signals are normally treated as CMOS levels, unless the board is operating in the GenStepper-only "1E" state ("Remote Direct Pulse Control"). This means that a logic "0" is generated at any time that the input voltage is <= ½ of the board power, and a logic "1" is generated when the input voltage is above ½ of the board power. Therefore, since our power is 5 volts, a logic "0" is presented when the input is <= 2.5 volts, and a "1" is presented when the signal is above 2.5 volts. In reality, we suggest using <=2 volts for a "0", and >=3 volts for a "1", to avoid any "noise" issues. When the board is in the "1E" state, then it changes to operating the TTL inputs as Schmitt-Triggered, to avoid false-step actions. In this case, the voltage levels require 0.6 volts for low, and 4.2 volts (or above) for high.

Note also that all of the TTL inputs are internally tied to +5 via 1K resistors. This permits you to use switch-closure-to-board-ground as your method of generating a "0" to the board, with the "1" being generated by opening the circuit.

Input Limit Sensors, lines LY- to LX+

Lines LY- through LX+ are used by the software to request that the motors stop moving when they reach a hardware-defined positional limit. Enabled by default at power on, the 'T' command may be optionally used to enable or disable any combination of these switches.

The connections are:

Signal	***Limit Sensed***
LY-	-Y
LY+	+Y
LX-	-X
LX+	+X

The connections may be implemented as momentary switch closures to ground; on the connector, a ground pin is available near the LY- pin. They are fully TTL compatible; therefore driving them from some detection circuit (such as an LED sensor) which generates TTL levels of signals will work. The lines are "pulled up" to +5V with a very weak (10-20K) resistor, internal to the SX-28 microcontroller. As an order option, the board also may be configured with on-board 1K pull-ups on all of the TTL input signals, to 'strengthen' the signals.

The stop requested by a limit switch normally is "soft"; that is to say, the motor will start ramping down to a stop once the limit is reached – it will **not** stop instantly at the limit point (unless a special firmware option is ordered). Note that if a very slow ramp rate is selected (such as changing the speed at only 1 microstep per second per second), it can take a very large number of steps to stop in extreme circumstances. It is quite important to know the distance (in microsteps) between limit switch actuation and the hard mechanical limit of each motorized axis, and to select the rate of stepping ("R"), rate of changing rates (the slope, "P"), and the stop rate ("K") appropriately.

As the most extreme example possible:

- if for some insane reason the motor is currently running at its maximum rate of 62,500 microsteps per second,
- and the allowed rate of change of speed is 1 microstep per second per second,
- and the stop rate was set to 1 microstep per second,
- then the total time to stop would be 62,500 seconds (a little over 17.3 hours -- groan!), with a distance of ½ v^2, or ½ (62,500)^2, or 1,953,125,000 microsteps.
- Note that this same amount of time would have been needed to get up to the 62,500 rate to begin with…

Therefore, it is strongly recommended that, if limit switch operation is to be used, these extremes be avoided. By default, the standard rate of change is initialized to 8000 microsteps/second/second, with the stop rate being set to 80 microsteps/second.

Also note that use of the "!" emergency reset command, or the "1E" followed by "0E" sequence (on GenStepper) will cause an immediate stop of the motor, regardless of any other actions or settings in the system. ***Please be aware that, in some designs, damage to gear systems can result when such a sudden stop occurs. Use this feature with care!***

Note that it is possible to order the firmware configured for "instant stop" on the limit switches. As with the '!' command, if the firmware is configured with this mode of operation, ***please be aware that, in some designs, damage to gear systems can result when such a sudden stop occurs. Use this feature with care!***

Motor Slew Control: Y- to RDY

Lines Y- through RDY are used to control stepping of the motors, and the rate of steps. The slew inputs are designed to operate via a microswitch closure to ground (or via TTL-driven logic levels).

The connections are:

Signal	***Action Requested***
Y-	-Y slew
Y+	+Y slew
X-	-X slew
X+	+X slew
POT	Changes Rate
RDY	Motors Ready

When operated normally, the indicated motor is "slewed" in the requested direction at the current rate, as long as the indicated signal is at ground level. Illegal combinations (such as Y- and Y+ both being low at the same time) are treated as "stop", to avoid confusion. As with all other operations of the system, each motor is accelerated to the current rate using the ramp rate defined within the code (which defaults to 4000 microsteps/second/second).

Under GenStepper, 'POT' input is treated as another TTL-input, wherein each time a closure occurs the code simply selects the "next" rate from its standard internal table of rates, and sets that rate as the requested rate for both motors. The standard rates currently provided after power on reset are:

- 16 microsteps (1 full step)/second
- 40 microsteps (2.5 full steps)/second
- 80 microsteps (5 full steps)/second
- 160 microsteps (10 full steps)/second
- 400 microsteps (25 full steps)/second
- 800 microsteps (50 full steps)/second (this is the power-on default)
- 1600 microsteps (100 full steps)/second
- 4000 microsteps (250 full steps)/second
- 8000 microsteps (500 full steps)/second

Under PotStepper and RelayStepper, the POT input may be attached to a user-provided potentiometer, which will be used by the code to set the rate (which can be overridden through use of the "R" command). Please refer to page 13 for more information on this.

Be forewarned that there is no way for the software to tell that a motor cannot operate at a given rate. On power-on, the default microstep is $1/16^{th}$ of a full step; therefore, the default rates range from 1 to 500 full steps/second. Changing the microstep size does change the above real "full step" rates – see the '!' command for more details.

The RDY output signal is used to indicate that motor motion is still being requested on at least one of the motors. When HIGH, then all motion is stopped. When LOW, at least one motor is still moving. This signal is LOW when the system is running under "remote pulse control" operation.

USB Driver Installation Under Windows for the SS0905 board

Our boards use a USB driver chip for communications with your hosting computer. FTDI (http://www.ftdichip.com) provides drivers for operation under Windows[tm], Linux, and Mac/OS. Our installation disk includes modified copies of their Windows[tm] drivers; our newest boards use a unique ID code which prevents them from being recognized by the default FTDI drivers. All Linux and Mac/OS customers must download their drivers directly from the http://www.ftdichip.com site, as we have no support capability for those platforms. They will also have to special-order the boards from us such that we configure them to respond to the standard FTDI drivers. Look for the drivers and documentation that relate to their FT232RL device.

A short summary of the installation of the drivers under Windows follows. For installation under Linux or on the Mac, please refer to the FTDI documentation available from their web site.

Base Driver Installation Under Windows

Installation of the drivers under Windows is fairly straightforward. If you are installing under Windows Vista[tm], you should read our more complete installation instructions as found in our "FirstUse" document. The key additions to the following list when installing under Vista are that you must be logged in as an administrator, and Vista will give you several extra verification prompts in order to confirm that you really want to do this (say 'Yes').

A short summary of the procedure under XP follows, along with a description of the adjustments that should be made to the COM emulator port settings after installation has been completed.

1. Thanks to the "magic" of "Plug-N-Play", connect the board to your computer (use a normal USB A-B cable of the appropriate length, connecting the 'A' side to your computer USB slot, and the 'B' side to our board). ***Make certain that the board is NOT on any sort of conductive surface (for example, metal, your hand, a carpet) when you do this, since you could damage the board (or your computer!) if any of the signals on the board get shorted.*** Note that you do NOT need to have the board connected to any external product (such as an actual motor driver) to install the drivers: just the board and a USB A-B cable are needed, in addition to your computer with its USB 1.1 or 2.0 connection.
2. If step 1 does not cause Windows to bring of their "Found New Hardware" wizard" or to otherwise recognize the board, then correctly power the board (see our manual section "Power Connector" on page 64 which identifies the power connector and describes the voltages which may be used).
3. This should cause Windows to bring up their "Found New Hardware" wizard, which will guide you through the installation process.
4. Place our installation CD into your CD drive.
5. If our setup application starts up, cancel out of it
6. Tell the wizard to "search for a suitable driver", and then tell it to "specify a location".
7. It will then ask for where to search: tell it to look in the "FtdiStepperBoard" directory on our support CD.
8. Then tell it to install the driver. If you are installing from the 'FtdiStepperBoard' version of the drivers, Windows will complain that the drivers are not 'Windows Certified'. You may ignore the error; all that is different between the 'ftdi' certified installation and the 'FtdiStepperBoard' non-certified installation is that the installation script sets the communication defaults to our recommended values (below) and adjusts the list of recognized devices to include our products.
9. The installation may then go through the same process in order to install the virtual COM drivers (if you have never installed an FTDI USB-based product before). Use the

same subdirectory and process to install those drivers as were used under step 7, above.

10. Once that process completes, the code will automatically add a new "COM" serial port which is "attached" to the board when it is plugged into the ***any*** USB port on your computer. *The system will automatically add a new COM port each time you attach a new board to any USB port on your computer or hub. It may also create a new COM port if you receive a repaired board back from us (if we have had to replace the USB driver chip).*

Initial testing of the board after driver installation – TestSerialPorts

The easiest way to test the board (and to identify which COM port is being used for board communications) is to run our "TestSerialPorts" application (found under 'StepperBoard' on your 'Start' menu). This application will scan all of the potential COM ports on your system (from COM1 through COM255), and will identify every port that has a connected StepperBoard product powered and attached.

The test assumes that the board is correctly configured to 'talk' to the com port: in the case of the A-BS0610 or A-BS0710 board using its on-board serial driver, the 'JS' jumper must be installed for the TestSerialPorts application to be able to locate the board.

When TestSerialPorts starts, simply press the "Scan Serial Ports" button (you may safely ignore the other buttons). The application will then perform its scan, and will identify every COM port on your system. It will also identify the baud rate for each connected board.

If TestSerialPorts does not locate your board, please contact us for additional tests to perform. Remember that the board must be connected to your computer and powered on, and the FTDI USB drivers must be correctly installed (for our USB based boards) for TestSerialPorts to be able to locate the board).

Please note that the TestSerialPorts application will locate our board even if you have not adjusted the default USB COM port properties, as described in the next section.

Adjusting Default COM port properties for best operation

Once the system has created the COM port for the board, you may need to change the system defaults to match the requirements of our motor controllers. If you installed from the 'FtdiStepperBoard' subdirectory, then these changes will normally have been done for you. Otherwise, you will need to perform the following procedure:

1. Under Windows 2000 or XP, go to your "system properties" page. Do this by
    a. Right-click on your "System" icon
    b. Select "Properties"
    c. Select "Hardware devices" (it might just be called "Hardware")
    d. Select "Device Manager"
2. Under Windows Vista, log in as an administrator, and then get to your "device manager" page by:
    a. Go to your 'Start' menu, and click on the 'Computer' button
    b. On the ribbon that appears at the top of the resulting window, click on "System Properties"
    c. On the task pane on the left of the new window, click on "Device Manager"
    d. The system will ask for your permission to continue. Press the "Continue" button.
3. Look under "Ports (COM and LPT)", and select the COM port that you just added (it will normally be the highest-numbered port on the system, such as "COM6"), and edit its properties. Note that the 'TestSerialPorts' application (described in the prior section) will have identified this COM port for you as part of its report.
4. Reset the default communication rate to:
    a. 9600 Baud,
    b. No Parity,
    c. 1 Stop Bit,
    d. 8 Data Bits,
    e. No Handshake
5. Select the "Advanced Properties" page, and set the:
    a. Read and Write buffer sizes to 64 (from their default of 4096).
    b. Latency Timer to 1 millisecond
    c. Minimum Read Timeout to 0
    d. Minimum Write Timeout to 0
    e. Serial Enumerator to checked
    f. Serial Printer to unchecked
    g. Cancel If Power Off to unchecked
    h. Event On Surprise Removal to unchecked
    i. Set RTS On Close to unchecked

(that is to say, only the Serial Enumerator is checked in the set of check boxes on the display)

Serial Operation

The RS-232 based serial control of the system allows for full access to all internal features of the system. It operates at 2400 (special order) or 9600 baud, no parity, and 1 stop bit. Any command may be directed to the X, Y or both motors; thus, each motor is fully independently controlled. Note that you should wait about ¼ second after power on or reset to send new commands to the controller; the system does some initialization processing which can cause it to miss serial characters during this "wake up" period.

Actual control of the stepper motors is performed independently for each motor. A "goto" mode is supported, as is a simple "go in a given direction". The code does support ramping of the stepping rate; however, it does NOT directly support changing the ramp rate, step rate, or "goto" target while a "goto" is under way. The behavior is either that the motor will *first* stop and *then* perform the new request, or that the new parameter value will be used on the *next* action. If button control is performed while a goto is underway, the goto gets changed to a direction slew, and the state of actions is reset.

Serial input either defines a new current value, or executes a command. The current value remains unchanged between commands; therefore, the same value may be sent to multiple commands, by merely specifying the value, then the list of commands. For example,

1000G

would mean "go to location 1000"

0G?

would mean "go to location 0, and while that operation was pending, do a diagnostic summary of all current parameters".

The firmware actually recognizes and responds each new command about ¼ of the way through the stop bit of the received character. This means that the command starts being processed about ¾ bit-intervals before completion of the character bit stream. In most designs, this will not be a problem; however, since all commands issue an '*' upon completion, and they can also (by default) issue a <CR><LF> pair before starting, it is quite possible to start receiving data pertaining to the command before the command has been fully sent! In microprocessor, non-buffering designs (such as with the Parallax, Inc.[tm] Basic Stamp [tm] series of boards), this can be a significant issue. The firmware handles this via a configurable option in the 'V' command. If enabled, the code will "send" a byte of no-data upon receipt of a new command character. This really means that the first data bit of a response to a command will not occur until at least 7-8 bit intervals after completion of transmission of the stop bit of that command (about 750 uSeconds at 9600 baud); for the Basic Stamp[tm] this is quite sufficient for it to switch from send mode to receive mode.

Serial Commands

The serial commands for the system are described in the following sections. The code is case-insensitive (i.e., "s" means the same thing as "S"). ***Please be aware that any time any new input character is received, any pending output (such as the standard "*" response to a prior command, or the more complex output from a report) is cancelled.*** This avoids loss of commands as they are being sent to the control board.

Serial Command Quick Summary

Most of the commands may be preceded with a number, to set the value for the command. If no value is given, then the last value seen is used.

0-9, +, - – Generate a new VALUE as the parameter for all FOLLOWING commands
A – Select the Auto-Full Power Step Rate
B – Select both motors
E – Enable or Disable Remote Direct Pulse Control (GenStepper firmware only)
G – Go to position x on the current motor(s)
H – Operate motors at ½ power
I – Wait for motor 'Idle'
K –Set the "Stop oK" rate
L – Latch Report: Report current latches, reset latches to 0
M – Mark location, or go to marked location
O – step mOde – How to update the motor windings
P – sloPe (number of steps/second that rate may change)
R – Set run Rate target speed for selected motor(s)
S – start Slew
T – limiT switch control (firmware versions 1.65 and above)
V – Verbose mode command synchronization
W – Set windings power levels on/off mode for selected motor
X – Select motor X
Y – Select motor Y
Z – Stop current motor
! – RESET – all values cleared, all motors set to "free", redefine microstep. Duplicates Power-On Conditions!
= – Define current position for the current motor to be 'x', stop the motor
? – Report status
other – Ignore, except as "complete value here"

0-9, +, - – Generate a new VALUE as the parameter for all FOLLOWING commands

Possible combinations:

"-" alone – Set '-' seen, set no value yet: used on SLEW -

"+" alone – Clear '-' seen, set no value yet: used on SLEW+

-n: Value is treated as -n

n: Value is treated as +n

+n: Value is treated as +n

Examples:

-s – Start slew in `-` direction on the current motor

-10s – Slew back 10 steps on the current motor

A – Select the Auto-Full Power Step Rate

This sets the approximate rate (expressed in the current microstep resolution; see the "!" command) at which the system automatically switches to full power to both windings, with strict full-step mode. This is used once the power loss induced by running at high speed becomes significant. *This mode will also disable ½ current mode ("1H") once this rate has been reached.*

Note that the code only stores the high byte of this value (i.e., the value divided by 256), and requires that the actual rate divided by 256 be above the value just set. This means that "A" rates of 0-255 all map into 0, and they set all rates 256 and above to be auto-full step mode. **The code defaults at power-on/reset to A=3072** ("3072a"). When the rate is "greater" than 3072, then the motor will run in the full-power, full-step mode. Observe that "A" values of 3072 through 3327 all generate the same test value! When operating at the default microstep resolution of $1/16^{th}$ step size, then the 3072 rate maps into 192 full steps/second. When operating at a microstep resolution of $1/64^{th}$ step size, then the same 3072 rate maps into 48 full steps/second.

For example,

3072A

would set automatic full-power mode to start when the microstep speed exceeds 3072 microsteps/second.

Set this to 62500 to disable this feature.

B – Select both motors

This command selects both the X and Y motors as targets for the following commands.

For example,

B0?

Would generate a report about all reportable parameters for both motors.

At power on/reset, both motors are selected for actions.

E – Enable or Disable Remote Direct Pulse Control (GenStepper Only)

This GenStepper firmware feature is used to control whether the TTL input lines are used as direct, edge triggered step requests for their associated motor and direction of travel. The current VALUE is used as the parameter. The options are bit encoded into the value; the low 2 bits of value define the main Pulse Control mode, the next 2 bits are extended feature selection, while the next higher 4 bits control the interpretation of the input signal level. The defined values for the low 2 bits (0-1) are:

0e – Disable remote pulse control (the power on/reset default). Note that the *entire* value must be 0 for remote pulse control to be disabled – if you define any of bits 2-7 to be non-0, then the code will act as if step mode '1' was selected.

1e – Enable remote pulse control, with each line being its own step/direction

2e – Enable "Step/Direction" mode of direct pulse control: the "-" inputs are treated as direction, the "+" inputs are treated as step requests.

As of firmware version 2.9, Bit 2 is used to define whether the limit switch inputs are used to control the current to the motors. If bit 2 is set (+4), then this extended TTL control of the motor current is enabled as described later in this section. If bit 2 is clear (+0), then the limit switches are either ignored or are used as true limits, depending on whether the firmware was ordered with the "hard stop" option.

Bit 3 is reserved for future expansion, and should be left as 0 at present.

Bits 4-7 are used to control the interpretation of the signal levels. When set to 0 (the default), then the signals are interpreted as described below. When set to 1, then the given signal is inverted (i.e., "0" is mapped into "1", and "1" is mapped into "0"). The net effect of this is to change the edge which triggers the motion from the low-going edge to the high-going edge, and to flip (when in mode 2) the interpretation of the direction of travel.

The bits are encoded as follows:

Bit	Value	Description
0-1	+0 to +2	Pulse mode to use: 0 means disable, 1 means each line is own step in its own direction, 2 means step-and-direction mode
2	+4	TTL control of motor current
3	*	reserved – leave 0
4	+16	Invert Y-
5	+32	Invert Y+
6	+64	Invert X-
7	+128	Invert X+

For example, to operate in the Step/Direction mode of operation, with high-going pulses requesting the steps on both the X and Y motors, you would use a value of 2+32+128 (since the Y+ and X+ input signals are used as the step requests, and need to be inverted so that the high-going edge triggers). Therefore, the command given would be "**162e**".

On both enable and disable, all pending motor actions are **immediately** stopped. The windings on both motors are forced on when remote pulse control is enabled, and are restored to the status defined by the W command when remote pulse control is disabled.

NOTE THAT:

- **THIS COMMAND IS FOR BOTH MOTORS**
- **IT IMMEDIATELY DISABLES ANY PENDING MOTIONS**
- **IF ANY MOTION IS UNDER WAY, THAT FACT IS FORGOTTEN. THIS CAUSES AN INSTANT STOP OF BOTH MOTORS! NO "GRADUAL STOP" (VIA THE AUTOMATIC RAMP MECHANISM) IS PERFORMED. MOTORS OR GEAR**

TRAINS MAY THUS BE DAMAGED IF THIS IS DONE IMPROPERLY ON SOME SYSTEMS.

- **TTL INPUTS FOR LIMIT SWITCHES ARE ALSO IGNORED DURING THIS MODE OF OPERATION, UNLESS THE 'HARD STOP' OPTION IS ORDERED.**
- **TTL INPUTS ARE TREATED AS "SCHMITT-TRIGGERED" DURING THIS MODE: <=0.4 Volts is "0", >=4.6 volts is "1".**

When enabled, then all other motor motion commands (such as G and S) have no effect (although changing the step mode, marking locations, and setting rates will affect the stored values for use when remote direct control is disabled). Instead, the TTL input lines are monitored frequently enough to sense 8 microsecond width pulses, looking for low-going-edges (leading edges) in the requests. The leading edges are then used to step the appropriate motor as needed. The stepping actions performed are always in units of the current microstep size, and are masked based on the current winding control rules (see the "!" command for how to control the microstep size, and the "O" command for control of winding/microstepping).

This mode monitors the TTL inputs very closely. It looks for leading (low-going) edges on each of the 4 TTL input lines (low means TRUE, high means FALSE, for compatibility with the normal switch mode of input), and issues a single microstep (in the current microstep precision). The rate of monitoring is such that, if pulses are 8 microseconds wide for each of the high and low states, they will be correctly sensed. Pulse widths less than 8 microseconds will usually be incorrectly processed! The effective maximum stepping rate is therefore 16 microseconds per microstep (both motors may be stepped at the same time), thus providing for a maximum step rate of 62,500 microsteps per second per motor. Since the maximum microstep rate is ½ full step per microstep, the maximum rate possible with this form of control is 31,250 full steps per second.

If mode 1 is used, then each input line ('x-', 'x+', 'y-', 'y+') is independently monitored for pulse edges, and is used to request a single step in the indicated direction.

If mode 2 is used, then each input line pair is used to control step and direction. 'x-' and 'y-' are used to determine the direction the indicated motor will spin on an associated step request (low means spin minus, high means spin plus). The 'x+' and 'y+' inputs are monitored for the related step requests: a low-going edge on the indicated line generates a step request on the associated motor. The restriction of timing is that each direction line ('x-' or 'y-') must be stable at least 20 ns before the low-going edge of the associated step line ('x+' or 'y+'), and must remain stable for at least 8 microseconds.

If the extra feature of "Limit switch control of the motor current" is requested (for example through the mode of "6E"), then the limit switches are interpreted as follows:

Limit Switch	Description
LY-	High means enable the Y motor, low means disable the Y motor (that is to say, if LY- is low, the Y motor is off)
LY+	If the Y motor is enabled (LY- is high), then LY+ controls the motor current used. High means use full current, low means use ½ current.
LX-	High means enable the X motor, low means disable the X motor (that is to say, if LX- is low, the X motor is off)
LX+	If the X motor is enabled (LX- is high), then LX+ controls the motor current used. High means use full current, low means use ½ current.

G – Go to position x on the current motor(s)

This is used to cause the currently selected motor(s) to travel to the indicated location (from the current Value). The software will:

- Calculate the direction and distance of travel
- Determine how long it has to 'ramp' the motor to go from its current start rate to the standard target rate
- Determine how long it has to then let the motor run at the target stepping rate
- Determine how long it will need to ramp the motor to stop it (which is the same time as that for starting the motor, above).
- Actually perform the action

The code ALWAYS starts from a stop, due to issues of timing. Therefore, if a "Goto" is performed while the motor is running, the system will first stop the motor (as in the 'Z' command), and then restart it based on its then-current location.

For example,

X1000gy-25687g

Would:

1. Select the X motor for actions
2. Start a GOTO on motor X to location 1000
3. Select motor Y for actions
4. Start a GOTO on motor Y to location –25687

Note that the two goto operations continue asynchronously until completed, unless a new command (such as a stop for that motor, or a change in direction request) is received. The current location for a given motor may always be requested, through the "-1" report. For example,

x-1?

Could report

X,-1,350

*

while the motor was still on its way to the requested location.

H – Operate motors at ½ power

"H" mode may be used to run a motor at a higher-than-rated voltage, in order to improve its torque. When 'H' is set to '1', then the PWM (Pulse Width Modified) count used to drive each winding is divided by two, thus cutting the effective current to the motor in half.

The two settings for this are:

0H – Run in normal FULL POWER mode (this is the normal power on/reset default)

1H – Run in ½ power mode

Note that if the "2W" mode is selected (for leaving windings on at ½ power when motion ceases), then the windings are actually left at ¼ power during idle. ***Please review the separate document "HalfPowerNotes.pdf" for a complete description of correct use of this capability.***

Note that, for firmware versions before 1.71, the board reverts to mode "0H" (full power mode) whenever it is reset (by a power cycle, low pulse on the RESET input line, or via the "!" command). If your code fails to detect this reset condition, you can cause a board failure by not re-issuing the "1h" command after a reset! Starting with firmware version 1.71, the code can automatically select the initial state of the ½ power mode by sensing the presence of a 1K resistor between the SO (Serial Output) signal and ground. If there is no resistor there, then full power mode will be selected (i.e., the same behavior as prior code versions). However, if there is a 1K resistor between SO and GND, then the firmware will select ½ power mode as the initial state, thus avoiding potential damage to the controller.

I – Wait for motor 'Idle'

This allows your code to 'wait' for the currently selected motor(s) to (both) be idle. The code simply waits for either the selected motors to have completed their motion (see the **X**, **Y**, and **B** commands) or for the next serial character to be received, and then it transmits the '*' prompt (ready for next command). Note that, if the wait is stopped by receipt of a new character, then the new character IS processed as part of a new command – it is NOT discarded.

For example, to go to a given X location, and then wait for the motor to actually get there, you could simply issue the command sequence:

Send	Receive
X	*
2000G	* *(note that the '*' is received as soon as the motion starts)*
I	* *(note that this '*' is not received until the motion completes)*

If you send a character before receipt of the final '*' (above), then system will discard transmitting the "*" response if it has not yet started the transmission. It will then process the new character. The best technique to avoid synchronization worries is to send two zero characters ('00'), wait for the second one to be completely sent, and then clear your input buffers. No further characters will be sent from the controller until it sees the next command after this 'flushing' action (i.e., any pending data transmissions will be aborted).

K –Set the "Stop oK" rate

This defines the rate at which the motors are considered to be "stopped" for the purposes of stopping or reversing directions. It defaults to the default of '80' if a value of 0 is given.

By default, this is preset to "80" upon startup of the system. This means that, whenever a stop is requested, the motor will be treated as "stopped" when its stepping rate is <= 80 microsteps (5 full steps) per second.

For example,

100k

sets the stop rates for the currently selected motor(s) to be 100 microsteps per second. Any time the current rate is less than or equal to 100, the motor will have the ability to stop instantly.

To set the rate such that the motors always immediately start and stop at the desired rate ('R') setting, issue the command:

62500K

This sets the 'Stop oK' rate to the maximum possible step rate, and thus will prevent all ramping behaviors of the code.

L – Latch Report: Report current latches, reset latches to 0

The "L"atch report allows capture of key short-term states, which may affect external program logic. It reports the "latched" values of system events, using a binary-encoded method. Once it has reported a given "event", it resets the latch for that event to 0, so that a new "L" command will only report new events since the last "L".

The latched events reported are as follows:

Bit	Value	Description
0	+1	Y- limit reached during a Y- step action
1	+2	Y+ limit reached during a Y+ step action
2	+4	X- limit reached during a X- step action
3	+8	X+ limit reached during a X+ step action
4	+16	System power-on or reset ("!") has occurred

For example, after initial power on,

```
L
```

Would report

```
L,16
*
```

If you were then to do an X seek in the "-" direction, and you hit an X limit, then the next "L" command could report:

```
L,4
*
```

M – Mark location, or go to marked location.

Based on the current parameter value (x), the M command will either cause the selected stepper(s) to record its'/their current position as the "marked" point, or will cause the location to be treated as a "goto" command.

x=0 : Mark current location for a later "go to mark" request

x=1 : Go to last "marked" location

O – step mOde – How to update the motor windings

The windings of the motors can be updated in one of three ways, depending on this step mode setting. By default, the code uses "micro step" mode set for 8 steps per complete full step, and performs a near-constant-torque calculation for positions between full step locations. The other modes include two full step modes and an alternating mode. For the full step modes, one enables only 1 winding at a time (low power), while the other enables 2 windings at a time (full power). The remaining mode alternates between 1 and 2 windings enabled.

The values which control this feature are:

- 0 : Full Step, Single winding mode (1/2 power full steps)
- 1 : Half step mode (alternate single/double windings on – non constant torque)
- 2 : Full step, double winding mode (full power full steps)
- **3 : Microstep, as fine as 1/8th step, constant-torque mode – This is the power on/reset default stepping mode.**

For example,

0o

sets the above ½ power full step mode, while

3o

sets the default microstep mode.

The "o" command does NOT affect the current step rates or locations; it only affects how the windings are updated. For example, when operating in the 1/8th step size, the following rules are applied for the various modes.

- 0: Single winding full step mode: Exactly one winding will be on at a time, and will be on at the selected current for the motor. The "real" physical motor position (in full step units) therefore only updates once every 8 microsteps; thus the "full step" location will be the (microstep location)/8, dropping the fractional part.
- 1: Half step mode: Alternates between having one and two windings on at a time, thus causing the torque to vary at the half-step locations. The "real" physical locations will be at half-step values, and hence the motor will "move" once every 3 microsteps. The "full step" location will be the (microstep location)/8, with fractions of 0 to 3/8 mapping into fractional location 0, and 4/8 though 7/8 mapping into fractional location 0.5.
- 2: Double winding full step mode: Both windings are "on" (at the selected motor current) at a time. As with mode 0, the "real" physical motor position will actually only update once every 8 microsteps. The "full step" location will be the (microstep location)/8, with the fractional part forced to 0.5.
- 3: Microstep mode. The current through the windings are precision-controlled, so that the microposition can be obtained. The physical motor position expressed in full step units is the (microstep location/8).

P – sloPe (number of steps/second that rate may change)

This command defines the maximum rate at which the selected motor's speed is increased and decreased. By providing a "slope", the system allows items which are connected to the motor to not be "jerked" suddenly, either on stopping or starting. In some circumstances, the top speed at which the motor will run will be increased by this capability; in all cases, stress will be lower on gear systems and motor assemblies.

The slope can be specified to be from 1 through 62,500 microsteps per second per second. If a value of 0 is specified, the code forces it to have a value of 8000. If a value above 62,500 (or less than 0) is specified, the code will accept it, but will ramp unreliably (i.e., do not do it!).

This value defaults at power-on or reset to 8000 microsteps per second per second. Please note that changing this during a "goto" action will cause the stop at the end of the goto to potentially be too sudden or too slow – it is better to first stop any "goto" in progress, and then change this slope rate.

For example, if we currently have motor X selected, and it is at location 0, then the sequence:

250p500r2000g

would cause the following actual ramp behaviors to occur:

1. The motor would start at its "stop oK" rate, such as 80 microsteps/second
2. It would accelerate to its target rate of 500 microsteps per second, at an acceleration rate of 250 microsteps/second/second.
3. This phase would last for approximately 500/250 or about 2 seconds, and would cover about 500 microsteps of distance.
4. It would then stay at the 500 microstep per second target rate until it was about 500 microsteps from its target location, i.e., at location 1500 (which would take another 2 seconds of time).
5. It would then slow down, again at a rate of 250 microsteps per second, until it reached the stop oK rate. As with the acceleration phase, this would take about 2 seconds.
6. The total distance traveled would be exactly 2000 microsteps, and the time would be 2+2+2=6 seconds (actually, very slightly less).

R – Set run Rate target speed for selected motor(s)

This defines the run-rate to be used for the currently selected motor(s). It may be specified to be between 1 and 62,500 microsteps per second; in this case, the requested rate becomes the target step rate for the motor(s). If a value of 0 is specified, the action depends on the firmware in use:

- GenStepper: the code forces a value of 400 to be used
- PotStepper or RelayStepper: the code returns control of the rate for the selected motor(s) to the potentiometer

If a value outside of the limits is specified, then it is accepted, but the code will not operate reliably. As with the ramP rate, do not specify values outside of the 1-62,500 legal domain unless you are returning rate control to the potentiometer (PotStepper or RelayStepper).

This defines the equivalent number of microsteps/second which are to be used to run the currently selected motor under the GoTo or Slew command. The internal motor position is updated at this rate, using a sampling interval of 62,500 update tests per second. The motor windings are then updated according to the stepping mode. For example, if the stepping mode (the 'o' command) for a given motor is one of the full-step modes instead of the microstep mode, and the microstep resolution is set to '1', then the motor will actually experience motion at 1/64th of the specified rate.

For example,

X250RY1000R

Sets the X motor target stepping rate to 250 microsteps per second, and the Y motor target rate to 1000 microsteps per second.

The power-on/reset default Rate is 800 microsteps/second.

If you are currently executing a targeted GoTo or Slew command which has a specific target location (i.e., "2000g" or "-300s"), the new rate will not take effect until the motion has completed. If you are executing a generic "Slew in a given direction" command ("+s" or "-s"), the new rate will take effect immediately, and the motor will change its rate to match the request using the current "P" (ramp-rate) value.

S – start Slew.

The "S"lew command is used to cause the currently selected motor to go in the selected direction. If the current value is only "+" or "-" (i.e., just has a sign associated with it), then the motor will slew in the indicated direction on the selected motor(s). Otherwise, the motor(s) will go VALUE steps in the direction indicated by the sign of VALUE, after first stopping the motor (more accurately, will target current location + x, then act as goto).

For example,

+s

will cause the current motor to start slewing in the forward direction, while

-250s

will invoke the "relative seek" calculation mode of the firmware.

When doing a relative seek (i.e., "-250s"), the address calculations are normally based on the current TARGET location, not the current instantaneous location. The actual rules are as follows:

1. If the given motor is currently executing a GoTo or relative Seek command, then the new location is calculated as a delta from the old target. For example,

```
Current State:
    Our current location is 1000
    Our current target is 2000
    We are doing a GoTo action
Request:
    -500s
Calculation:
    Since we are doing a normal GoTo,
    the new target location will be "2000-500", or 1500
Result:
    Motor stops, then goes forward to location 1500
```

2. Otherwise, the current location is treated as the value to calculate from for the relative motion. For example,

```
Current State:
    Our current location is 1000
    We are executing a "+s" command (slew positive)
Request:
    -500s
Calculation:
    Since we are executing a Slew,
    the new target location will be "1000-500", or 500
Result:
    Motor stops, then goes backward to location 500
```

This was set up this way as being a reasonable compromise on the intent of the meaning of "relative". If you want to force the motion to be strictly relative to the current location, you issue the "z" (stop) command first. Once that has been issued, the motor is placed in a special state (stopping, no target), which permits relative slew to be from the current location.

For example, to go -500 steps from the current location, regardless of whether the current action is a slew or a targeted goto, issue the command:

z-500s

T – limiT switch control (firmware versions 1.65 and above)

The limi'T' switch command is used to control interpretation of the board limit switch input. **By default (after power on and after any reset action), the board is configured to respond to each of the four limit switches; that is to say, all of the limit switches are enabled.** Control of this feature allows the board to more easily control rotary tables, which may only have an "index" switch instead of a "left and right limit" switch.

Please note that this capability was introduced in firmware version 1.65. It is not available in earlier releases of the firmware.

The command takes a bit-encoded parameter, which lists which switches are to be blocked from action. ***Note that in version 1.80, the feature of control of the sense levels for the limit switches was added***. The values are:

Bit	Numeric Sum Value	Action
0	+1	Block Y-
1	+2	Block Y+
2	+4	Block X-
3	+8	Block X+
4	+16	Sense level, LY-
5	+32	Sense level, LY+
6	+64	Sense level, LX-
7	+128	Sense level, LX+
All other bits		Reserved – do not use

Note that bits 4-7 (limit switch sense level) are ignored on versions of the firmware before 1.80. For version 1.80 and later, those bits are used to define the input level for the indicated limit input lines which are used to stop motor motion. A 0 means "use a logic low to stop", while a 1 means "use a logic high to stop". By default, the system uses a logic low to stop, so that the inputs (which are internally pulled high) will not cause a motor to stop if they are not connected.

For example,

4t

would block detection of the "X-" limit, and allow all of the other limits to work as normal.

240t

would invert the sense of all of the limit input sensors, so that a low means "operate" and a high means "limit reached".

V – Verbose mode command synchronization

The 'V'erbose command is used to control whether the board transmits a "<CR><LF>" sequence before it processes a command, and whether a spacing delay is needed before any command response. **By default (after power on and after any reset action), the board is configured to echo a carriage-return, line-feed sequence to the host as soon as it recognizes that an incoming character is not part of a numeric value.** This allows host code to fully recognize that a command is being processed; receipt of the <LF> tells it that the command has started, while receipt of the final "*" states that the command has completed processing.

The firmware actually recognizes and responds each new command about ½ of the way through the stop bit of the received character. This means that the command starts being processed about ½ bit-interval before completion of the character bit stream. In most designs, this will not be a problem; however, since all commands issue an '*' upon completion, and they can also (by default) issue a <CR><LF> pair before starting, it is quite possible to start receiving data pertaining to the command before the command has been fully sent! In microprocessor, non-buffering designs (such as with the Parallax, Inc.[tm] Basic Stamp [tm] series of boards), this can be a significant issue. All firmware versions 1.54 and above handle this via a configurable option in the 'V' command. If enabled, the code will "send" a byte of no-data upon receipt of a new command character. This really means that the first data bit of a response to a command will not occur until at least 9 bit intervals after completion of transmission of the stop bit of that command (about 900 uSeconds at 9600 baud); for the Basic Stamp[tm] this is quite sufficient for it to switch from send mode to receive mode. *The firmware also adds 2 additional "stop" bits to each transmitted character, when this feature is enabled. This is to allow non-buffering microprocessors some additional time to do real-time input processing of the data.*

The verbose command is bit-encoded as follows:

Bit	SumValue	Use When Set
0	+1	Send <CR><LF> at start of processing a new command
1	+2	Delay about 1 character time before transmission of first character of any command response. On firmware versions 1.60 and later, add 2 more stop bits to each transmitted character, to allow more processing time in the receiving microprocessor.

If you set verbose mode to 0, then the <CR><LF> sequence is not sent. Reports still will have their embedded <CR><LF> between lines of responses; however, the initial <CR><LF> which states that the command has started processing will not occur.

For example,

0v

would block transmission of the <CR><LF> command synch, and could respond before completion of the last bit of the command, while

3v

would enable transmission of the <CR><LF>sequence, preceeded by a 1-character delay. The complete table of options is:

Value	Delay First	<CR><LF>
0	No	No
1	No	Yes
2	Yes	No
3	Yes	Yes

W – Set windings power levels on/off mode for selected motor

The "W"indings command controls whether the currently selected motor(s) has its windings left enabled or disabled once any GoTo or Slew action has completed, and it controls power levels to use during normal stepping. It is acted on immediately – that is to say, if the current motor(s) is (are) stopped, then the windings are *immediately* engaged or disengaged as requested.

The values to use for control are:

0w – Full power during steps, completely off when stepping completed (default setting)

1w – Full power at all times (both during steps and when idle)

2w – Full power during steps, 50% power when idle

This mode is used to reduce power consumption for the system. When windings are disengaged, they draw very little power; however, their full rated power is drawn when they are engaged. If windings are off, then the stepper motor will "relax", and will move on its own to a "preferred location", controlled by its fixed magnets (thus inducing up to ½ step's worth of positional error). If they are on, the motor is actively held at its requested location (and the motor itself heats up). If mode 2 is used (the 50% power setting), then the windings are pulsed at about ½ of the normal rate, thus the power requirements are ½ of the normal amount for the given location, after a goto or slew has completed.

X – Select motor X

This command selects X motor as the target for the following commands.

For example,

X100r

Would cause the step rate to be set to 100 for motor X.

Y – Select motor Y

This command selects Y motor as the target for the following commands.

For example,

Y100r

Would cause the step rate to be set to 100 for motor Y.

Note that if the controller is operating in "**single motor dual power**" mode (selected by grounding both LY- and LY+ during power on/reset operations), then any commands sent to the Y motor controller are effectively ignored. Only the X motor controller sends signals to the X and Y connectors when that mode is enabled.

Z – Stop current motor.

'Z' causes the current motor(s) to be ramped to a complete stop, according to its current ramp rate and stepping rate. "Stopped" is defined as "having a step rate which is <= the stop oK rate". See the 'K' command for defining the "stop oK rate".

For example,

Xz

Would slow down, then stop motor X.

! – RESET – all values cleared, all motors set to "free", redefine microstep. Duplicates Power-On Conditions!

This command acts like a power-on reset. It ***IMMEDIATELY*** stops both motors, and clears all values back to their power on defaults. No ramping of any form is done – the stop is immediate, and the motors are left in their "windings disabled" state. This can be used as an emergency stop, although all location information will be lost.

The value passed is used as the new microstep size, in fixed $1/64^{th}$ of a full step units. **At raw power on, the board acts like a "4!" has been requested**; that is to say, it sets the microstep size to 4x1/64, which is $1/16^{th}$ of a full step. By issuing the '!' command, you can redefine the microstep size to a value convenient for your application. The value must range from 1 to 32; it is clipped to this range if exceeded.

The suggested values would be the powers of 2, vis. 1, 2, 4, 8, 16, 32 and 64 (giving you true microstep step sizes of 1/64, 1/32, 1/16, 1/8, ¼, ½ and 1 respectively). All other values (such as RATE or GOTO LOCATION) are then expressed in units of the microstep size; therefore, location "3" would mean "3/64" in the finest resolution (microstep set to 1), and "3" in the largest resolution (microstep set to 64). Note that the ability to specify 64 started with version 1.75; all earlier versions had an upper limit of $32/64^{th}$ of a step (1/2 step) as the largest step size.

For example,

4!

resets the system to its power on default of 1/16 microstep resolution.

The reset command also selects the following settings:

- **3072A** – Set the Automatic Full Step rate to be >=3072 microsteps/second
- **B** – Select both motors for the following actions
- **0**= – Reset both motors to be at location 0
- **0H** – Set motors to full power mode
- **80K** – Set the "Stop OK" rate to 80 microsteps/second
- **3O** – Set the motor windings Order to "microstep"
- **8000P** – Set the rate of changing the motor speed to 8000 microsteps/second/second
- **800R** – Set the target run rate for the faster motor to 800 microsteps/second
- **0T** – Enable all limit switch detection
- **1V** – Set <CR><LF> sent at start of new command, no transmission delay time
- **0W** – Full power to motor windings

= – *Define current position for the current motor to be 'x', stop the motor*

This copies the current VALUE as the current position for the selected motors, and then stops said motor(s). For example,

X2000=Y4000=

Would define the current location of the X motor to be 2000, and the current location of the Y motor to be 4000. Note that no actual motor motion is involved – the code simply defines the current location to be that found in the VALUE register, and issues an automatic stop ('Z') request. Note that the motor is stopped AFTER the assignment is complete, so the actual "current position" of the motor will be different from this value, depending on how long it takes for the motor to stop.

X2000=g

Would define the current location of the X motor to be 2000, and then would actually go to that 2000 location. This combination could be used when the motor is actually slewing or executing a "goto", to force the "current" location to be set and selected.

? – Report status

The "Report Status" command ("?") can be used to extract detailed information about the status of either motor, or about internal states of the software.

For a status report, the value is interpreted as from one of three groups:

1-255: Report memory location 1-255

Useful locations: ***NOTE THAT ANY LOCATION ABOVE 7 MAY CHANGE BETWEEN CODE VERSIONS***

- 5: Port A register – this contains the limit switches
- 6: Port B register – this contains the TTL inputs
- 7: Port C register – this controls the motor windings

0: Report all of the following special reports except for version/copyright

-1 to -12: Do selected one of the following reports

- -1; Report current location
- -2; Report current speed
- -3; Report current slope
- -4; report target position
- -5; Report target speed
- -6; Report windings state
- -7; report stop windings state
- -8; Report step action (i.e., motor state)
- -9; Report step style
- -10; Report run rate
- -11; Report stop rate
- -12; Report current software version and copyright
- other: Treat as 0 (report all except version/copyright)

All of the reports follow a common format, of:

1. If Verbose Mode is on, then a <carriage return><line feed> ("crlf") pair is sent.
2. The letter corresponding to the motor being reported on is sent (i.e., 'X' or 'Y').
3. A comma is sent.
4. The report number is sent (such as –4, for target position).
5. Another comma is sent.
6. The requested value is reported.
7. If this is a report for both the X and the Y motors, then a <crlf> is sent.
8. If this is a report for both motors, the other report is sent.
9. If Verbose Mode is on, then a <crlf> is sent
10. A "*" character is sent.

If both motors are being reported, a line containing the X report is sent, followed by a line containing the Y report.

Finally, a "*" character is sent, which notifies the caller that the report is complete.

Note that in the following examples, first line of "Received" is "*". This is because two commands are actually being sent (i.e., "B", then "-<whatever>?"), and each command always generates a "*" response once it has been completed. Technically, fully "synchronized" serial communication consists of (1) send a command, and (2) save all characters until the "*" response is seen. The intervening characters are the results of the command, although only report ("?") and reset ("!") generate any significant response.

The special reports which are understood are as follows:

0: Report all reportable items

The "report all reportable items" mode reports the data as a comma separated list of values, for reports –1 through –11. Just after power on, for example, the request of "0?" would generate the report:

X,0,a,b,c,d,e,f,g,h,I,j,k,l,m

Where:

- X is the motor: such as 'X' or 'Y'
- 0 is the report number; 0 is the 'all' report
- a is the value for the current location (report "-1")
- b is the value for the current speed (report "-2")
- c is the value for the current slope (report "-3")
- d is the value for the target position (report "-4")
- e is the value for the target speed (report "-5")
- f is the value for the windings state (report "-6")
- g is the value for the stop windings state (report "-7")
- h is the value for the step action (motor state) (report "-8")
- i is the value for the step style (both full step modes and half) (report "-9")
- j is the run rate (report "-10")
- k is the stop rate (report "-11")

For example,

```
B0?
```

Would report all reportable values for both motors. You could receive:

```
*
X,0,30,10,1000,30,10,0,0,0,1,100,10
Y,0,-300,10,1000,-300,10,0,0,0,1,100,10
*
```

-1: Report current location

This reports the current (instantaneous) location for the selected motor(s).

For example,

```
B-1?
```

Would report the current location on both motors. You could receive:

```
*
X,-1,10
Y,-1,25443
*
```

-2: Report current speed

This reports the current (instantaneous) speed for the selected motor(s).

For example,

```
B-2?
```

Would report the current speed on both motors. You could receive:

```
*
```

```
X,-2,800
Y,-2,2502
*
```

-3: Report current slope

This reports the current (instantaneous) rate of changing the speed for the selected motor(s).

For example,

```
B-3?
```

Would report the current rate on both motors. You could receive:

```
*
X,-3,10
Y,-3,25443
*
```

-4: Report target position

This reports the target location for the selected motor(s).

For example,

```
B-4?
```

Would report the current target on both motors. You could receive:

```
*
X,-4,100
Y,-4,-35443
*
```

-5: Report target speed

This reports the current target run rate which is desired for the selected motor(s). This value is usually either the current stop rate (we are attempting to slow down to this speed) or the current requested run rate (as reported by –10, and as requested by the 'R' command) depending on whether we are speeding up or slowing down.

For example,

```
B-5?
```

Would report the target rate on both motors. You could receive:

```
*
X,-5,800
Y,-5,250
*
```

-6: Report windings state

This reports the current energized or de-energized state for the windings for the selected motor(s). A reported value of 0 means "the windings are off", a value of 1 means "the windings are energized in some fashion".

For example,

```
B-6?
```

Would report the current state on both motors. You could receive:

```
*
X,-6,1
Y,-6,0
*
```

-7: Report stop windings state

This reports whether the windings will be left energized when motion completes for selected motor(s). A reported value of 0 means "the windings will be turned off", a reported value of 1 means "the windings will be left at least partway on".

For example,

```
B-1?
```

Would report the requested state on both motors. You could receive:

```
*
X,-1,1
Y,-1,0
*
```

-8: Report current step action (i.e., motor state)

This reports the current (instantaneous) state for the selected motor(s). The step action may be one of the following values:

- 0: Idle; all motion complete
- 1: Ramping up to the target speed, in a "GoTo"
- 2: Running at the target speed, in a "GoTo"
- 3: Slowing down, from a "GoTo"
- 4: Slewing ("+s")
- 5: Quick stop in progress ("z", or saw a limit switch closure)
- 6: Reversing direction
- 7: Stopping in preparation for a new GoTo
- 8: Single shot: current action finished (you probably will never see this; it is only selected for about 8 uSeconds)

For example,

```
B-8?
```

Would report the current location on both motors. You could receive:

```
*
X,-8,0
Y,-8,4
*
```

This would mean that motor X is idle, while motor Y is currently doing some form of slew operation.

-9: Report step style (i.e., micro step, half, full)

This reports the current method of stepping for the selected motor(s). The legal step styles reported are those of the "O" (step mode) command, vis:

- 0: Full step, single windings
- 1: Half step, alternating single/double windings
- 2: Full step, double windings
- 3: Microstep
- +4 added to above: **Single Motor Dual Power** mode is enabled (i.e., both LY- and LY+ were grounded during the last power on/reset of the controller).

For example,

```
B-9?
```

Would report the current stepping method on both motors. You could receive:

```
*
X,-9,3
Y,-9,2
*
```

This would equate to the X motor being in microstep mode, while the Y motor is running in full-power, full step mode.

If you were connected in **dual power mode**, then you could get a report such as:

```
*
X,-9,7
Y,-9,6
*
```

Even though a mode will be reported for the Y motor controller, it is actually ignored in terms of sending signals to the Y motor connector; only the X motor controller affects the signals sent to the X and Y connectors when in dual power mode.

-10: Report run rate

This reports the current requested run rate for the selected motor(s). This is the last value set by the "R" command.

For example,

```
B-10?
```

Would report the current rate on both motors. You could receive:

```
*
X,-10,2000
Y,-10,3200
*
```

-11: Report stop rate

This reports the speed at which the motors may be considered to be stopped, for starting and stopping activities for the selected motor(s).

For example,

```
B-11?
```

Would report the current stop rate on both motors. You could receive:

```
*
X,-11,80
Y,-11,50
*
```

-12: Report current software version and copyright

This reports the software version and copyright.

For example,

```
B-12?
```

could report:

```
*
genstepper.src $version: 1.48$
Copyright 2002 by Peter Norberg Consulting, Inc. All Rights
Reserved
*
```

other – Ignore, except as "complete value here"

Any illegal command is simply ignored, other than sending a response of "*". However, if a numeric input was under way, that value will be treated as complete. For example,

123 456G

would actually request a "GoTo location 456". Since the " " command is illegal, it is ignored; however, it terminates interpretation of the number which had been started as 123.

Note that, upon completion of ANY command (including the 'ignored' commands), the board sends the <carriage return><line feed> pair, followed by the "*" character.

More Examples

For example,

Y 1000 R

Would set the Y rate to 1000 steps/second. The spaces are optional, and would not prevent the code from working; however, an extra "<cr><lf>*" sequence would be sent by the board for each space seen.

B50R

Would set both the Y and X rates to 50 steps per second

300YG

Would go to Y location 300

800G

Would go to location 800 on the most recent motor (in this example, Y)

Y-S

Would start slewing in the minus direction on Y motor

Y+SX3S

Would start slewing positive on Y motor, and would go + 3 steps on the X motor

X1SSS

Would step forward 3 steps on the Y motor, since the calculation is based on the CURRENT TARGET location at the time of the command if the motor is currently executing a GOTO or relative step slew, and is otherwise based on the current MOTOR location. This is thus exactly equivalent to

X3s

X100rY300RB0g

Would cause the step rate to be set to 100 for motor X, 300 for motor Y, and then cause both motors to go to location 0.

Direct TTL Step Control – GenStepper Firmware Only

The 'E' command as supported by the GenStepper firmware (see the 'E' command under 'Serial Control') allows a remote controller (another microprocessor, another computer, etc.) to directly request microsteps going in either direction on either (or both) stepper motor(s). The step size used is the current microstep size and is masked based on the current winding control rules (see the "!" command for how to control the microstep size, and the "O" command for control of winding/microstepping). The sampling rate is such that at most 62,500 microsteps/second may be requested on each motor.

NOTE THAT:

- **THIS COMMAND IS FOR BOTH MOTORS**
- **IT IMMEDIATELY DISABLES ANY PENDING MOTIONS**
- **IF ANY MOTION IS UNDER WAY, THAT FACT IS FORGOTTEN. THIS CAUSES AN INSTANT STOP OF BOTH MOTORS! NO "GRADUAL STOP" (VIA THE AUTOMATIC RAMP MECHANISM) IS PERFORMED. MOTORS OR GEAR TRAINS MAY THUS BE DAMAGED IF THIS IS DONE IMPROPERLY ON SOME SYSTEMS.**
- **TTL INPUTS FOR LIMIT SWITCHES ARE NORMALLY IGNORED DURING THIS MODE OF OPERATION, UNLESS SPECIAL FIRMWARE OPTIONS ARE ORDERED OR SPECIAL CURRENT CONTROL REQUESTS ARE MADE.**

The TTL input lines which are normally used to request a "slew" of a motor in a given direction (when low) get redefined to request a "step" of a motor in a given direction when going low. The wiring thus is:

Signal	***Action Requested***
Y-	-Y microstep
Y+	+Y microstep
X-	-X microstep
X+	+X microstep

The code samples the above lines at a rate such that the minimum time low and minimum time high for each pulse is 8 microseconds (each); shorter pulses may be missed.

A standard sequence to use pulse-based control of the system would thus be:

1. Make certain that the TTL inputs (Y- through X+) are all high.
2. Set up the base microstep size as needed (for example, to step at the maximum precision, issue a "1!" to reset the controller to 1/64 step).
3. Wait about 1/2 second for the reset to complete.
4. Issue the correct winding control command, if needed (by default, the system operates in mode "3o", which is the microstep mode).
5. Issue the "1E" command, to enable TTL based remote control.
6. From now on, until the "0E" (or reset) is issued, a "leading-edge-to-zero" state change on any of the 4 TTL input lines will request a step in the direction of that line.
    - For example, bringing "Y+" low (for at least 5 microseconds) will request a positive (micro) step on the Y motor. The Y+ line must then be brought back high,

for at least 5 microseconds, before a new request is guaranteed to be recognized on that line.

- A motion may be requested at the same time on both the X and Y motors; illegal combinations (such as Y- and Y+ both requesting a step at the same time) are ignored.
- Note that there is no upper limit on how wide this pulse may be; it just has to be no narrower than 5 microseconds in each direction.

Serial operations which do not request a change in the state of the motor may be processed while running in the TTL mode of control without loss of pulses or steps; however, doing commands which change state may cause lost TTL pulses on inputs and skewing of the PWM signal on outputs.

The following commands will cause up to 16 microseconds of missed TTL control edges during their processing (hence one or two pulses can theoretically be missed). Due to the fact that they are only of use when not in remote TTL control mode, they should not be used in that mode.

- 'G' – GoTo
- 'I' – wait for motor Idle (during remote TTL control mode, this command never completes)
- 'M' – Mark location
- 'P' – sloPe rate
- 'R' – target Rate
- 'S' – start Slew
- 'Z' – stop
- 'W' – winding mode when stopped (windings are always ON in TTL mode)

The following commands will also cause up to 16 microseconds of missed TTL control edges, and should therefore be used with care. However, they do affect the behavior of the system when in remote TTL control mode, and hence may be of use.

- 'H' – Half power
- 'O' – step mOde
- '=' – set location
- '!' – Reset the controller; abort all actions, restart system.

All of the other commands may be used with no negative effects on timing in the system.

Basic Stamp™ Sample Code

The StepperBoard series of boards may all be used with the Parallax, Inc.tm Basic Stamp tm series of boards. The connection to the board is usually via three of the pins on the IO connector: RDY, SI, and SO, with the JS jumper removed from the board. The remaining input pins on the IO connector may be wired or not, as needed by the application. Most of the time, they will be left unconnected (to "float").

Communications between the two boards is normally be performed at 9600 baud (the default). 2400 baud may be used as a special-order option. Normally, operating at the 9600 baud rate is recommended; use the 2400 baud rate only if you cannot make your code work at 9600 baud. You **must** use the 'V'erbose command to configure the controller to pause one character time before sending responses to the Basic Stamp, to avoid data synchronization issues.

The sample code provided by Peter Norberg Consulting, Inc. assumes that the following connections have been made between the StepperBoard and the Basic Stamp:

- RDY connected to P3
- SI connected to P2
- SO connected to P1

Some of the code provided operates at 2400 baud. Note that, in reality, all of the code can run correctly at 9600 baud on most stamps; operation at 2400 baud is shown here just to demonstrate the technique.

"Gendemo.bs2" is a 9600 baud demo, which uses the READY line for synchronization. It runs using a microstep size of 4/64 (1/16) of a full-step, and constantly spins both motors between logical position 2000 and 0. On each "spin cycle", the stepping mode gets changed; each of the legal stepping modes (full step 2-winding, full step 1-winding, ½ step, and microstep) are exercised in sequence, and a 1/5 of a second pause is inserted between each cycle for ease of visual synchronization.

"Gendemoser.bs2" is a 9600 baud demo, which ignores the "READY" line and uses the SERIAL input line for all of its synchronization. Aside from operating strictly using the serial communications interface, it operates identically to "Gendemo.bs2".

"Genseekser.bs2" is a somewhat more comprehensive example, in terms of showing the capabilities of the StepperBoard system. As with "Gendemoser.bs2", this operates at 9600 baud. It operates at the full level of microstep possible (1/64 of a full step), and runs each motor at a different speed. X is set to a maximum rate of 4000 microsteps/second (which is 4000/64 or 62.5 full steps/second), with a matching ramp rate of 4000 microsteps/second/second. Y is set to a maximum rate of 8000 microsteps/second (which is 125 full steps/second), with a ramp rate of 7000 microsteps/second/second. It also sets the "automatic full-power" step rate to be 6000 microsteps/second. Given that only Y will exceed this rate, the Y motor will switch from what ever mode it is using to full power mode during any seek which goes far enough for it to exceed the 6000 microsteps/second rate. Having gone through this setup, the loop operates similarly to that in "Gendemoser.bs2", except that the locations cycled are +16,000 and 0. If you use this demo with two identical motors, you should be able to "hear" the difference in the stepping modes, and you should also hear the Y motor "become noisy" partway through the microstep phase of the entire sequence (when it switches between microstep mode and full power full step mode).

The complete sources to these examples are installed by default into the "C:\StepperBoard" directory, when you install the code provided with the product.

Listing for GENDEMO.BS2 – 9600 Baud, READY line based

```
' ****************************************************************************
' $modname: gendemo.bs2$
' $nokeywords$
' Demonstrates some of the serial commands using goto and TTL Busy line to the
' SimStep and BiStep
' set of controllers from Peter Norberg Consulting, Inc.
'
' The tool first initializes the stepper to operate at 16 microsteps/full step,
' with the start/stop rate being 80 uSteps/second, and the ramp rate at
' 1000 uSteps/sec/sec.
' The target ramp rate is 1000 uSteps/second;
' The auto-power switch mode (the 'A' command) is left at its default of 3072,
' which is equivalent to 192 full
' steps/second.
'
' Note that both motors are selected for the actions by default.
' It then enters the speed test loop.
'
' The code first waits for the stepper unit to report idle.
' and it is instructed to move to logical location 2000 (in) 1/16th steps.
' (Note that this is full step location 62.5).
' This is then followed by a move to location 0, and then a new stepping mode
' is selected.  A 1/5th second pause is inserted to make it easy to identify
' when the cycle is occurring.  All three modes of stepping are cycled:
'   Mode    Use
'    0      Single Winding mode (1/2 power full steps)
'    1      Half step mode (alternate single/double windings on)
'    2      Full step mode (double windings on)
'    3      Microstep mode (full microstep processing; DEFAULT MODE)
'
'
' SPECIAL TIMING NOTE: It can take the SimStep/BiStep up to 100 uSeconds to respond to
' a new serial "go" command (goto or slew); therefore, you must always wait
' a small amount of time (at least a few milliseconds uSecs) before testing the
' "busy" line, since
' you may get a "false idle" response.
'
' Additional note: The SimStep/BiStep products operate at 9600 baud.  Although
' the Basic Stamp series can send this rate reliably,  many of them cannot receive
' at this rate without data loss; therefore, no attempt is made in this
' sample to receive serial data from the controller.
' ****************************************************************************

' {$STAMP BS2}


' SimStep or BiStep connected as follows
'   Serial Input P1 to SimStep B7 Serial output
'   Serial Output p2 to SimStep B6 Serial Input
'   busy          p3 to SimStep B5 Status Output
'                  (HIGH = idle, LOW = motion in progress)
'                  AND busy NOT connected to 1K resistor to ground (force 9600 baud)

PortStepperSerFrom  con  1      ' Serial from stepper port
PortStepperSerTo    con  2      ' Serial to stepper port
PortStepperBusy     con  3      ' Busy line
PortStepperBaud     con 84      ' Baud rate to generate 9600 baud:
                                ' Must have no pull-down resistor on busy line!

PortStepperBusyTest var in3    ' Same as PortStepperBusy, used for input test

idMicroStep var byte            ' Gets microstep mode; cycles 0 to 3


' Code restarts here if RESET button pressed

        input             PortStepperBusy         ' BUSY from stepper

        pause 250                                  ' Wait for stepper power on cycle
        serout PortStepperSerTo,PortStepperBaud,["4!"]
                          ' Reset the stepper, set 4/64 full-step step size
```

```
        pause 1000
                  ' Wait for stepper to send its wake-up copyright text
        serout PortStepperSerTo,PortStepperBaud,["80K"]
                  ' Set Stop OK to 'can start/stop at 80 microsteps/sec'
        serout PortStepperSerTo,PortStepperBaud,["1000p"]
                  ' For demo purposes, set a slow ramp of 1000 microsteps/sec
        serout PortStepperSerTo,PortStepperBaud,["1000R"]
                  ' For demo purposes, set a target rate of 1000 microsteps/sec
        idMicroStep = 0
                  ' Start at microstep 0


loop:
        serout PortStepperSerTo,PortStepperBaud,[dec idMicroStep,"o"]
                  ' Set microstep mode

        serout PortStepperSerTo,PortStepperBaud,["2000g"]
                  ' Go to location 2000

        gosub WaitReady
                  ' Wait until ready

        serout PortStepperSerTo,PortStepperBaud,["0g"]
                  ' Go back to 0

        idMicroStep = (idMicroStep + 1) & 3
                  ' Cycle step type
        gosub WaitReady                     ' Wait until ready
        pause 200                           ' wait 0.2 seconds before we cycle
        goto        loop                    ' Cycle forever

WaitReady:
        pause 100    ' Wait 0.1 seconds for prior character to be processed
        if PortStepperBusyTest = 0 then WaitReady       'Wait till not busy
        return
```

Listing for GENDEMOSER.BS2 – 9600 baud, serial based

```
' ****************************************************************************
' $modname: gendemoser.bs2$
' $nokeywords$
' Demonstrates some of the serial commands using goto and serial response to
' the SimStep and BiStep
' set of controllers from Peter Norberg Consulting, Inc.
'
' The tool first initializes the stepper to operate at 16 microsteps/full step,
' with the start/stop rate being 80 uSteps/second, and the ramp rate at
' 1000 uSteps/sec/sec.
' The target ramp rate is 1000 uSteps/second;
' The auto-power switch mode (the 'A' command) is left at its default of 3072,
' which is equivalent to 192 full
' steps/second.
'
' Note that both motors are selected for the actions by default.
' It then enters the speed test loop.
'
' The code first waits for the stepper unit to report idle.
' and it is instructed to move to logical location 2000 (in) 1/16th steps.
' (Note that this is full step location 62.5).
' This is then followed by a move to location 0, and then a new stepping mode
' is selected.  A 1/5th second pause is inserted to make it easy to identify
' when the cycle is occurring.  All three modes of stepping are cycled:
'   Mode    Use
'    0      Single Winding mode (1/2 power full steps)
'    1      Half step mode (alternate single/double windings on)
'    2      Full step mode (double windings on)
'    3      Microstep mode (full microstep processing; DEFAULT MODE)
'
'
' SPECIAL TIMING NOTE: It can take the SimStep/BiStep up to 100 uSeconds to respond to
' a new serial "go" command (goto or slew); therefore, you must always wait
' a small amount of time (at least a few milliseconds uSecs) before testing the "busy"
' line, since
' you may get a "false idle" response.
'
' Additional note: The SimStep/BiStep products normally operate at 9600 baud.
' Although the Basic Stamp series can send this rate reliably,  many of them
' cannot receive at this rate without data loss; therefore, a special patch has
' been made available to the GenStepper versions 1.75 and later, to allow for
' 'slowing down' of the response to a command.  By issuing the '2V' command, the
' code will wait one complete character time (about 1 millisecond) before sending a
' response; this gives enough time for the stamp to reset for serial input.
' ****************************************************************************

' {$STAMP BS2}


' SimStep or BiStep connected as follows
'   Serial Input P1 to SimStep B7 Serial output
'   Serial Output p2 to SimStep B6 Serial Input
'   busy          p3 to SimStep B5 Status Output
'                    (HIGH = idle, LOW = motion in progress)
'                    AND busy connected to 1K resistor to ground (force 2400 baud)

PortStepperSerFrom  con  1      ' Serial from stepper port
PortStepperSerTo    con  2      ' Serial to stepper port
PortStepperBusy     con  3      ' Busy line
PortStepperBaud     con 84      ' Baud rate to generate 9600 baud

PortStepperBusyTest var in3     ' Same as PortStepperBusy, used for input test

idMicroStep var byte            ' Gets microstep mode; cycles 0 to 3
'szSerString var byte(2)        ' Only used if you enable debug mode (see comments)


' Code restarts here if RESET button pressed

       input             PortStepperBusy               ' BUSY from stepper
```

```
        pause 250
           ' Wait for stepper power on cycle
        serout PortStepperSerTo,PortStepperBaud,["4!"]
           ' Reset the stepper, set 4/64 full-step step size
        pause 1000
           ' Wait for stepper to send its wake-up copyright text
        serout PortStepperSerTo,PortStepperBaud,["2V"]
           ' Set short responses, but add delay before response
        serout PortStepperSerTo,PortStepperBaud,["80K"]
           ' Set Stop OK to 'can start/stop at 80 microsteps/sec'
        serout PortStepperSerTo,PortStepperBaud,["1000p"]
           ' For demo purposes, set a slow ramp of 1000 microsteps/sec
        serout PortStepperSerTo,PortStepperBaud,["1000R"]
           ' For demo purposes, set a target rate of 1000 microsteps/sec
        idMicroStep = 0
           ' Start at microstep 0


loop:
        serout PortStepperSerTo,PortStepperBaud,[dec idMicroStep,"o"]
           ' Set microstep mode

        serout PortStepperSerTo,PortStepperBaud,["2000g"]
           ' Go to location 2000

        gosub WaitReady                                    ' Wait until ready

        serout PortStepperSerTo,PortStepperBaud,["0g"]     ' Go back to 0

        idMicroStep = (idMicroStep + 1) & 3 ' Cycle step type
        gosub WaitReady                                    ' Wait until ready
        pause 200                          ' wait 0.2 seconds before we cycle
        goto        loop                   ' Cycle forever

WaitReady:
'        DEBUG "Waiting..."
        serout PortStepperSerTo,PortStepperBaud,["00I"]
           ' wait for ready; the leading 0's flush BiStep's output queue

       SerIn PortStepperSerFrom,PortStepperBaud,[WAIT("*")]    ' And wait for done
'       SerIn PortStepperSerFrom,PortStepperBaud,[STR szSerString\1]
'     DEBUG "Saw: ", STR szSerString, "[", HEX szSerString(0), "]", CR
        return
```

Listing for GENSEEKSER.BS2 – 9600 Baud, serial based, complex actions

```
' ************************************************************************
' $modname: genseekser.bs2$
' $nokeywords$
' Demonstrates some of the serial commands using seek and serial response
' to the SimStep and BiStep
' set of controllers from Peter Norberg Consulting, Inc.
'
' The tool first initializes the stepper to operate as follows:
'   64 microsteps/full step,
'   start/stop rate being 320 uSteps/second
'   ramp rate at 4000 uSteps/sec/sec for the X motor, 7000 uSteps/second for the
'   Y motor.
'   Auto-power switch mode (the 'A' command) is reset to 6000 uSteps/second
'   Target ramp rate is 4000 uSteps/second for X, 8000 uSteps/second for Y
'
' This combination means that the X motor will peak at 1/2 the speed of the Y motor,
' and that the Y motor will switch to full-step full power mode during the midpoint
' of the seek.
' During the microstep pass test (when idMicroStep = 3), you will notice that the
' Y motor
' will start quietly, and then suddenly become noisy for a short period, and then
' it will quiet
' down again.  This is occurring when the stepping mode switches from micro to
' full when
' the motor speed is faster than about 6000 uSteps per second.
'
'
' Note that both motors are selected for the seek actions.
' It then enters the speed test loop.
'
' The code first waits for the stepper unit to report idle.
' and it is instructed to move +16000 (in) 1/64th steps.
' (Note that this is full step delta 125).
' This is then followed by a move to location -16000, and then a new stepping mode
' is selected.  A 1/5th second pause is inserted to make it easy to identify
' when the cycle is occurring.  All three modes of stepping are cycled:
'   Mode    Use
'    0      Single Winding mode (1/2 power full steps)
'    1      Half step mode (alternate single/double windings on)
'    2      Full step mode (double windings on)
'    3      Microstep mode (full microstep processing; DEFAULT MODE)
'
'
' SPECIAL TIMING NOTE: It can take the SimStep/BiStep up to 100 uSeconds to respond to
' a new serial "go" command (goto or slew); therefore, you must always wait
' a small amount of time (at least a few milliseconds uSecs) before testing the
' "busy" line, since
' you may get a "false idle" response.
'
' Additional note: The SimStep/BiStep products normally operate at 9600 baud.
' Although the Basic Stamp series can send this rate reliably,  many of them
' cannot receive at this rate without data loss; therefore, a special patch has
' been made available to the GenStepper versions 1.75 and later, to allow for
' 'slowing down' of the response to a command.  By issuing the '2V' command, the
' code will wait one complete character time (about 1 millisecond) before sending a
' response; this gives enough time for the stamp to reset for serial input.
'
' Since this is a relative seek on both motors, you can test the limit switches
' easily;
' just ground one of the limit inputs (A0-A3) at a time, and observe which motor stops
' going
' which direction.
'
'       Ground  Direction
'       Line    Blocked
'       A0      -Y
'       A1      +Y
'       A2      -X
'       A3      +X
'
' ************************************************************************
```

```
' {$STAMP BS2}


' SimStep or BiStep connected as follows
'   Serial Input P1 to SimStep B7 Serial output
'   Serial Output p2 to SimStep B6 Serial Input
'   busy          p3 to SimStep B5 Status Output
'                  (HIGH = idle, LOW = motion in progress)
'                  AND busy connected to 1K resistor to ground (force 2400 baud)

PortStepperSerFrom  con  1      ' Serial from stepper port
PortStepperSerTo    con  2      ' Serial to stepper port
PortStepperBusy     con  3      ' Busy line
PortStepperBaud     con 84      ' Baud rate to generate 9600 baud:

PortStepperBusyTest var in3    ' Same as PortStepperBusy, used for input test

idMicroStep var byte            ' Gets microstep mode; cycles 0 to 3
'szSerString var byte(2)        ' Only used if you enable debug mode (see comments)


' Code restarts here if RESET button pressed

        input             PortStepperBusy               ' BUSY from stepper
        pause 250
           ' Wait for stepper power on cycle
        serout PortStepperSerTo,PortStepperBaud,["1!"]
           ' Reset the stepper, set 1/64 full-step step size
        pause 1000
           ' Wait for stepper to send its wake-up copyright text
        serout PortStepperSerTo,PortStepperBaud,["2V"]
           ' Set short responses, but add delay before response
       serout PortStepperSerTo,PortStepperBaud,["320K"]
           ' Set Stop OK to 'can start/stop at 320 microsteps/sec'
        serout PortStepperSerTo,PortStepperBaud,["6000A"]
           ' Set auto-switch to full power mode to 6000 microsteps/sec;
           ' only Y will do it
        serout PortStepperSerTo,PortStepperBaud,["X"]
           ' For demo purposes, Select just X for a moment
        serout PortStepperSerTo,PortStepperBaud,["4000p"]
           ' For demo purposes, set X slow ramp of 4000 microsteps/sec
        serout PortStepperSerTo,PortStepperBaud,["4000R"]
           ' For demo purposes, set X target rate of 4000 microsteps/sec
        serout PortStepperSerTo,PortStepperBaud,["Y"]
           ' For demo purposes, Select just Y for a moment
        serout PortStepperSerTo,PortStepperBaud,["7000p"]
           ' For demo purposes, set Y faster ramp of 7000 microsteps/sec
        serout PortStepperSerTo,PortStepperBaud,["8000R"]
           ' For demo purposes, set Y target rate of 8000 microsteps/sec
        serout PortStepperSerTo,PortStepperBaud,["B"]
           ' For demo purposes, Select both X and Y for remaining actions
        idMicroStep = 0
           ' Start at microstep 0


loop:
        serout PortStepperSerTo,PortStepperBaud,[dec idMicroStep,"o"]
           ' Set microstep mode

        serout PortStepperSerTo,PortStepperBaud,["+16000s"]
           ' Go forward 16000 (real "full step" loc = 16000/64 = 250)

        gosub WaitReady                                    ' Wait until ready

        serout PortStepperSerTo,PortStepperBaud,["-16000s"]
           ' Go back to 0

        idMicroStep = (idMicroStep + 1) & 3 ' Cycle step type
        gosub WaitReady                                    ' Wait until ready
        pause 200                          ' wait 0.2 seconds before we cycle
        goto        loop                   ' Cycle forever

WaitReady:
```

```
'        DEBUG "Waiting..."
        serout PortStepperSerTo,PortStepperBaud,["00I"]
           ' wait for ready; the leading 0's flush BiStep's output queue


       SerIn PortStepperSerFrom,PortStepperBaud,[WAIT("*")]    ' And wait for done
'        SerIn PortStepperSerFrom,PortStepperBaud,[STR szSerString\1]
'     DEBUG "Saw: ", STR szSerString, "[", HEX szSerString(0), "]", CR
       return
```

SerTest.exe – Command line control of stepper motors

The SerTest.exe application (provided as part of the sample software) is a simple tool which allows command line based control of the StepperBoard product line (the SimStep and BiStep boards). It allows a batch-based script to control stepper motors, with no further need for any programming knowledge. All sources are provided, to allow rewriting as needed.

SerTest allows you to send command strings and see their responses, by issuing commands from the command prompt window. It is called as

```
SerTest Text1 Text2 … Textn
```

Where "Text1", "Text2", … are the actual strings to send to the controller (as described in the "Serial Commands" section of this manual). The code supports extended control of its behavior, by parsing the first character of each space-separated parameter on the command line – if it starts with "/", then the rest of that parameter is interpreted as a command to SerTest, instead of being sent to the controller. The commands recognized by SerTest are:

- /b#### Set Baud rate to ####; defaults to /b9600

  For example,

  /b9600 sets 9600 baud,

  /b2400 sets 2400 baud. No other values are useful.

- /i#### Set Idle wait time to #### milliseconds; defaults to /i60000

  The "Idle wait time" is the maximum amount of time (in milliseconds) which the software waits before it decides that a command has timed out, and thus that it is time to send the next command. This is used to maintain correct synchronization of the code with the controller. For example,

  /i60000 – Set 1 minute before timeout

  /i10000 – set 10 seconds before timeout

- /pCOMn set the serial communications port to port n; defaults to /pCOM1

  This allows control of which serial port is used for the following commands. The code does not actually attempt open any serial port until the first real data is ready to be sent to the controller; thus no attempt will be made to access COM1 if the command line looks like:

  SerTest /pCOM2 4!x1000g

  Note that if multiple /p commands are on the line, the most recent one seen is the one used at any given time. It is legal to have one command line actually operate multiple controllers!

All other text is passed, unchanged, to the controller. SerTest is aware of the general command structure for the StepperBoard product line; thus, it will correctly wait for synchronization each time a complete command is sent. All data received by SerTest is echoed back to the command prompt, thus allowing the operator to see the response to any command (or set of commands).

For example,

Sertest 4!x1000gy-2000gi

Would:

1. Operate at 9600 baud on COM1 using a 1 minute time out
2. reset the board to operate with a microstep size of 4/64
3. tell the X motor to go to location 1000,
4. tell the y motor to go to location -2000,

5. and wait up to 60 seconds for the motions to complete

Similarly,

SerTest /pCOM2 /b2400 /i10000 y+5000s

Would:

1. Operate using port COM2 at 2400 baud, with a timeout of 10 seconds
2. Tell the Y motor to seek forward 5000 steps

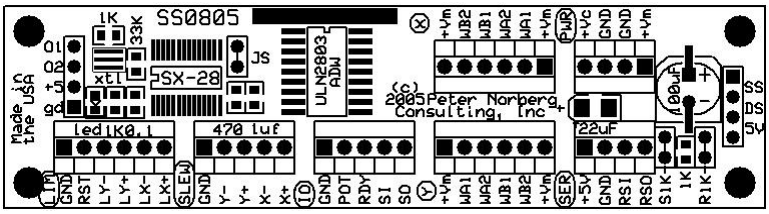
StepperBoard.dll – An ActiveX controller for StepperBoard products

The StepperBoard.dll object is a fairly comprehensive sample Visual Basic COM/ActiveX application which allows any COM-aware system (such as VBScript based scripts) to easily control the StepperBoard products. As with the SerTest application, all sources are provided, so that the user may change the system as needed.

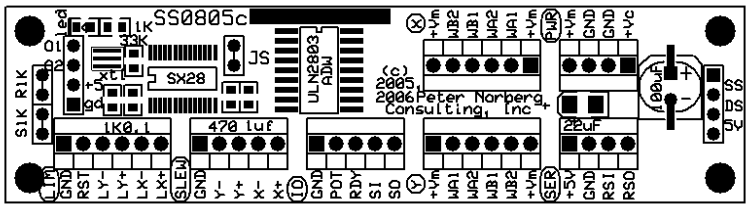
The program is well-documented in the manual StepperBoardClass.pdf. Please refer to that manual for more information about the product.

Board Connections

The current SS0805 beta board silkscreen is as shown as follows.

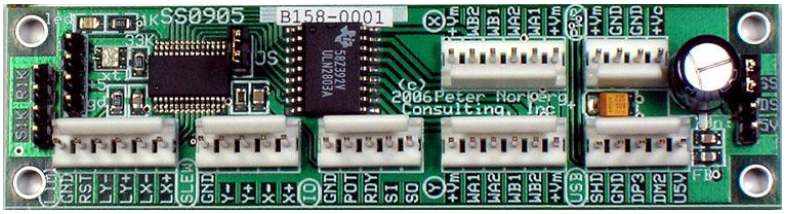


The expected release version SS0805a board silkscreen is as shown as follows.



The key difference between the two is the power connector (top right connector, labeled PWR). The '+Vm' and '+Vc' signals were swapped on the beta artwork; the final artwork corrects them so as to match the standard 3.5" floppy connector pinouts. Also, the R1K and S1K jumpers were moved to the left-hand side of the board, so that the layout would match the SS0905 board.

Similarly, the release version of the SS0905 appears as follows:



Board Size

The board, oriented as shown on this page, is 1.0 inch high by 3.8 inches wide.

Mounting Requirements

The board may be mounted using four machine screws, of sizes #2 through #5 (the holes are 0.125 inches in diameter). The holes are positioned exactly 0.125 inches in from each corner.

Connector Signal Pinouts

There are eight connectors on each board.

Going from top-left down in a counter-clockwise direction, we have:

- SX-Key debugger connector (4 pin SIP header)
- S1K and R1K jumpers
- TTL Limit Input and RESET (GND, RST, LY- to LX+)
- TTL Motor Direction Slew Control (Y- to X+)
- Board status and TTL Serial (POT, RDY, SI (serial input), SO (serial output)). Only use the TTL serial if the JS jumper, located near the SX-28 chip, is removed.
- Y motor connector
- RS232 (SS0805) or USB (SS0905) serial connector
- Power Option jumpers
- Power connector
- X motor connector

SX-Key debugger connector

Pin	Name	Description
1	GND	Vss (gnd) for SX-Key
2	+5V	+5 for SX-Key
3	OSC2	Oscillator Input 2
4	OSC1	Oscillator Input 1

This connector allows use of the Parallax, Inc.[tm] SX-Key debugger/programmer product, to reprogram the SX-28 in place. ***If the SX-Key is used as a debugging device, then the resonator (XTL) MUST BE REMOVED, or damage to the SX-Key may occur!***

TTL Limit Input and Reset

Name	Description
GND	Ground reference for inputs – short input to GND to denote limit
RST	Reset the microcontroller, when low
LY-	Y Minimum limit reached, when low
LY+	Y Maximum limit reached, when low
LX-	X Minimum limit reached, when low
LX+	X Maximum limit reached, when low

This connector is used to warn the SX-28 that a limit is being reached in the given direction. The signals are designed to be shorted to GND to denote that a limit is present; they are also compatible with normal TTL outputs from any TTL compatible device. LY- through LX+ are internally pulled up to +5 with 10K resistors (within the SX-28 itself).

TTL Motor Direction Slew Control

Name	Description
GND	Signal ground
Y-	Slew Y Negative
Y+	Slew Y Positive
X-	Slew X Negative
X+	Slew X Positive

This connector gives access to the TTL motor direction control signals for the system.

Y- through X+ are inputs, used to control manual slew requests. They each cause the indicated motor to turn at its current rate in the indicated direction, as long as the indicated signal is grounded. For example, connecting pin Y- to GND (or providing a low TTL input signal) will cause the Y motor to go in the "negative" direction.

Board status and TTL Serial

Name	Description
GND	Ground reference for all signals
POT	Rate control
RDY	Ready/busy output
SI	INPUT: Raw SX-28 Serial Input (TTL level)
SO	OUTPUT: Raw SX-28 Serial Output (TTL Level)

This connector gives access to the serial control signals for the SX-28, as well as board status and slew rates.

POT is used to select the rate, as described elsewhere in this manual.

RDY is normally an informational output that describes the state of "one or more motors are still stepping". High means READY/IDLE, low means STEPPING. **Do NOT use this to directly run a significant current (such as an LED):** during power on, this is sampled as an input to configure double current mode (the R1K jumper).

SI and SO are the "real" serial input and serial output (respectively), as seen by the SX-28 chip. If the application desires direct serial communications without RS232 levels (for example, if the Parallax Inc.[tm] Basic Stamp [tm] based products are being used to control the board), simply remove the JS jumper (the jumper is near the SX28 processor chip) and use these pins.

Note that if this board is a "child" board in a SerRoute controlled tree of boards, then the JS jumper will normally be removed, unless special interfacing is done.

The communication rate is fixed at order time to operate at 2400 or 9600 baud, no parity, 8 data bits, 1 stop bit. Normally, it is configured for 9600 baud operation.

RS232 Serial (SS0805)

Name	Description
+5V	Access to board +5 power
GND	Signal Ground
RSI	RS232 Serial input to the board (from the external device). You normally connect this to pin 3 of your DB9 cable.
RSO	RS232 Serial Output from the board (to the external device). You normally connect this to pin 2 of your DB9 cable.

This connector provides for all external serial communications, using the RS232-C standard.

USB Serial (SS0905)

Name	Description
SHD	USB Shield
GND	USB Signal Ground (USB pin 4)
DP3	USB Data signal plus (USB pin 3)
DM2	USB Data signal minus (USB pin 2)
U5V	USB +5V (USB pin 1)

This connector provides for all external serial communications, using the USB standard. Note that the portion of the board which supports the USB communications is powered by the USB +5V and USB Signal Ground pins; that is to say, as soon as the USB cable is connected between the board and the computer, and power is applied to the cable, then the USB portion of the SS0905 is enabled. This does not include enabling of the rest of the board, however! All that is enabled is that the computer can recognize that there is a USB device present.

Power Connector

Name	Description
+Vm	5-26 volts, for the X and Y motors
GND	Ground for Vm
GND	Ground for Vc
+Vc	+6.5-15 volts for the logic circuits, or 5.0 volts if the on-board regulator is being bypassed

The power connector has two sets of power and ground pins. This is mainly to make it possible to deliver power to high-voltage motors (i.e., any motor which needs more than 15 volts) while still powering the logic circuit off of its required 6.5 to 15 or 5 volts. Your power supplies must be regulated DC power supplies, of the appropriate current and voltage to operate the board and the motors – see the next section (starting on page 65) for information which should help you make your selection.

There are several ways of powering the system, which are dependent upon the current and voltage requirements of the system. One or two power supplies may be used, depending upon the voltage needed by the motors and upon whether extra cooling can be applied to the 2904 voltage regulator. If the motors require more than 15 volts to operate, **using the same power supply to the 2904 will cause the 2904 to get extremely hot** (over 100 deg. C). Although it technically can withstand temperatures up to 150 deg. C, we do not recommend or warrant it. It is much better in this case to split the supplies. Use GND and +Vc to provide 6.5 to 15 or 5 volts (if the power jumper is set to 5VO) to the 2904 (the lower voltage you use, the better it is from a heat point of view) at 200 mA. Use pins +Vm and GND to provide the motor power in this case.

Always select the correct power on the Power Option jumper block which is near the power connector – otherwise you will destroy your board!

The power options can thus be summarized as:

Motor Voltage	**Power Jumper Setting**	**Use sep. Power supply**	**Comments**
6.5-15V	SS	NO	Single power supply, connected to pins GND and +Vm of power connector. The SS jumper is installed on the power options header
15-26V, logic supply is 6.5 to 15 volts	DS	YES	Use two power supplies, one for the motor (connected to pins GND and +Vm), the other for the digital power (pins GND and +Vc). The digital power should be 6.5 to 15 volts, at least 200 ma. The DS jumper installed is installed on the power options header
15-26V, logic supply is 5 volts	5V	YES	Use two power supplies, one for the motor (connected to pins GND and +Vm), the other for the digital power (pins GND and +Vc). The digital power must be 5.0 volts, at least 200 ma. The 5V jumper installed on the power options header.

Calculating Current And Voltage Power Supply Requirements

This section note describes how to calculate the power requirements for your motors, and for the system as a whole.

1. Determine the individual motor winding current requirements.

The first issue is to determine the individual winding current requirements for your stepper motor. Since our system does not monitor current at all (it only estimates current, using a PWM-like technique), the current ratings as seen by our board may not match those specified by a manufacturer who is assuming that current-monitoring based control is being performed.

From the point of view of determining the current requirements for your motor, our system is best modeled using the standard resistor-only based formula (ignoring inductance) of:

V=IR

or, rearranging terms in order to find I,

I=V/R

That is to say, the current (I) as seen by our board equals the voltage (V) from your power supply divided by the resistance (R) of your motor windings. This value can be much greater than that claimed by a given motor manufacturer, since most of them assume that you are using a current-controlled system to run their motors.

For example, if you have a 3 ohm resistance in your windings, then the motor will "draw" 6/3 or 2 amps if 6 volts is driven out of it, and it will draw 12/3 or 4 amps (per winding!) if 12 volts is generated.

2. Determine current requirement for actually operating the motor(s)

Once you have determined the motor current, then you will need to determine how you intend to run it via our product offerings. We have four modes of operation, which provide for three levels of power per motor. These modes are controlled by the "o" command, which specifies the technique used to drive the windings.

Update Order	***Absolute Current Multiplier***	***Recommended Current Multiplier***
0 (single winding full step)	1.0	1.4
1 (half step: alternate 1, 2 windings)	2.0	2.5
2 (full step: 2 windings at a time)	2.0	2.5
3 (microstep)	1.7	2.3

Note that the "Recommended Current Multiplier" column in the above table includes a "fudge factor"; we always recommend using a power supply which is somewhat larger than the absolute minimum required, in order to avoid overloading issues.

Obviously, if you are going to run multiple motors off of one supply, you will need to add together all of the currents needed in order to determine how large of a supply to use.

For example, if you are going to microstep (mode 3) a motor whose winding current has been calculated to be 0.4 amps, then your power supply needs to be able to supply 2 x 0.4, or 0.8 amps to drive that particular motor.

4. Note the logic supply requirements

The current needed by the logic portion of the SS0805 board is 200 mA.

5. Determine the power supplies you will be using

Your choices are dependent on the desired voltage to the motors, and on the board which you have purchased from us. In all cases, we strongly recommend that linear regulated supplies be used: regulated switching supplies are often not very good when used with inductance based loads.

The logic supply must either be in the voltage range of 6.5 through 15 volts, or be 5 volts; use the power options jumper to 'tell' the board about your selection. If your motor voltage requirements are outside of this range, then you will have to use a split supply (as described below).

Single Supply.

If your motor power supply voltage is from 6.5 to 15 volts, then you may choose to use a single supply to operate the system. The current capabilities of the supply must exceed the sum of the current requirements of the motor(s) and the logic circuits. In this case, power the board using the '+Vm' and "GND" power connectors, and set the power options jumper to the "SS" position.

Dual Supply

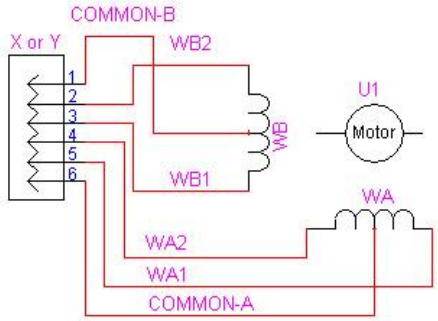
You may choose to separate the motor supply from the logic supply. If you do so, we suggest using the lowest voltage in the range of 6.5 to 15 volts on the logic supply which you have available, to reduce generation of waste heat on the board.

The motor supply should be from 5 to 26 volts, and otherwise is as calculated under sections 1 and 2, above. If the supply is to drive 2 motors, please remember to double the current needs.

You will use the +Vm and nearest GND connector to power the motors (from your motor power supply), and the +Vc and its nearest ground to power the logic. You will set the power options jumper to "DS" if your logic supply is in the range of 6.5 to 15 volts, or "5V" if it is a 5 volt supply.

Wiring Your Motor

There are two identical connectors used to operate the X and Y motors. The connectors are labeled with respect to which motor they operate. (This designation affects only which commands are to be used to control the motors; no other functionality is changed.) They are wired as follows for the SS0805 series of controllers:



Typical Unipolar Motor Connection
To the BiStepA05 or SimStepA04

Pin	Name	Description
1	+Vm	+Vm
2	WB2	Winding B, pin 2
3	WB1	Winding B, pin 1
4	WA2	Winding A, pin 2
5	WA1	Winding A, pin 1
6	+Vm	+Vm

Stepping sequence, testing your connection

The current is run through these connectors to generate a clockwise sequence as follows:

Step	***WB2***	***WB1***	***WA2***	***WA1***
0	0	0	0	**1**
1	0	**1**	0	**1**
2	0	**1**	0	0
3	0	**1**	**1**	0
4	0	0	**1**	0
5	**1**	0	**1**	0
6	**1**	0	0	0
7	**1**	0	0	**1**

Note also that it is explicitly legal when using the GenStepper firmware to operate your motor in "double current, ½ power mode". In "double current" mode, you wire your motor to both the X and Y motor connectors, and you jumper the board as described in the 'Configuring Double Current' mode section of this manual. In all other respects, you follow the rest of the instructions in this manual.

The actual wiring configuration to connect to a given stepper motor depends on the motor type. For most unipolar motors, each winding has three leads. The center-tap (shown in the above schematics as "COMMON-A" or "COMMON-B") is connected to +Vm on the SS0805 series of controllers. The other two leads are connected to pins WA-1 and WA-2 or WB-1 and WB-2, as shown in the above schematics. For bipolar motors, the windings match the labels – that is to say, pins 2-3 are for winding B, and 4-5 are for winding A. Note that the unipolar motors will also match the labels, but it may be more difficult to identify the windings.

Determining Lead Winding Wire Pairs

If there is no manufacturer's wiring diagram available, unipolar and bipolar motor windings can both often be identified with an ohm-meter by performing tests of their resistances between the motor leads.

For any motor, number the leads (from 1 to 4 for a bipolar motor, from 1 to 5 or 6 for a unipolar motor). Then measure the resistances and record the values in the empty cells in a table like the following:

	1	**2**	**3**	**4**	**5**	**6**
1	-					
2	-	-				
3	-	-	-			
4	-	-	-	-		
5	-	-	-	-	-	
6	-	-	-	-	-	-

For example, the cell at location (1,2) would be filled in with the resistance between leads 1 and 2. The '-' entries show values which do not need to be separately measured, since they are already measured in another row/column pair (or are a self-reading). For example, having measured the resistance between leads 1 and 2 to fill in cell (1,2), there is no reason to separately measure leads 2 and 1! If you have fewer leads than those shown in the table, ignore the rows and columns with the nonexistent leads.

For a 4-wire bipolar motor, the low-resistance pairs are the opposite ends of matching windings; high-resistance pairs are different windings. For example, if cell (1,2) shows 10 ohms, while (1,3) shows greater than 1000 ohms, then wires 1 and 2 can be called winding A, while wires 3 and 4 can be called winding B.

For a 5-wire unipolar motor, you will observe 2 reading values in the resulting table, with the higher reading being about double that of the lower reading. The single line which has the lower reading on all of its entries in the table is the common lead; the other wires are the winding leads (unfortunately, this test cannot show which is winding A and which is winding B through resistances alone).

For a 6-wire unipolar motor, you will observe 3 reading values in the resulting table.

- If you see a single reading near 0, then the two leads associated with that reading are the common leads, and the remaining 4 wires are the windings WA1, WA2, WB1 and WB2 (this test cannot determine which is winding A or B through resistances alone). As a check, you can observe that all readings between the other wires and either of the 2 common wires have value ½ that of all of the readings between the non-common wires.
- Otherwise, you will see readings which are near infinity (which identify leads from different windings), are at some value (such as 10), or are at double that value (such as 20). The pairs which show the "double value" are the opposite ends of a given winding (i.e., WA1 and WA2, or WB1 and WB2). The remaining wires are the "common" leads for their given windings.

A 6-wire 4-phase unipolar motor will have two "common" wires. You will normally connect one of the wires to pin 1, and the other to pin 6.

Sequence Testing

Always double check all of your power and motor connections before you apply power to the system. If you have reversed any power leads, you will blow out our board and you may blow out your power supply! If you are operating a unipolar motor and you short a common lead to a winding pin (WA or WB), then you will blow out our drivers! Similarly, any winding which is shorted to any other winding may burn out our board. If you are setting up to use double-power mode (connecting one motor to both the X and Y motor connectors in order to drive a larger motor), failure to follow the instructions in the 'Configuring Double Current Mode' section of this manual will also cause the board to fail. None of these issues are warranted failures; repairs for such are not covered!

After winding lines have been determined, identifying a running sequence can be done by testing the lines using following sequence, connecting to the X motor with clip leads. ***Turn off power*** to the board in between each test, so that power is not on while you change the wiring.

For wires A, B, C, and D (where A, B, C, and D are initially connected to the WA1, WA2, WB1, and WB2 lines) try these orders:

	WA1	***WA2***	***WB1***	***WB2***
1.	A	B	C	D
2.	A	B	D	C
3.	A	D	B	C
4.	A	D	C	B
5.	A	C	D	B
6.	A	C	B	D

For each pattern, request a motor motion in each direction using the applicable technique:

- GenStepper/PotStepper Firmware, using a terminal emulator:
    - Issue the command "x1000gi0gi", which should cause the motor to spin to logical location 1000, then back to 0. Wait for the "*" response after each sub-command (the "h", "x", "g", and "i" commands) before typing the next command, in order to let the firmware finish processing the request.
- NCStepper Firmware, using a terminal emulator:
    - Issue the command "1000xg0xgi", which should cause the motor to spin to logical location 1000, then back to 0. Wait for the "*" response after each sub-command (the "h", "x", "g", and "i" commands) before typing the next command, in order to let the firmware finish processing the request.

Only when the motor is wired correctly will you get smooth motion first in one direction and then the other.

Once a possible pattern has been determined, you may find that the direction of rotation is reversed from that desired. To reverse the rotation direction, you can either turn the connector around (this may be the easiest method, if a SIP style connector is used), or you can swap both the WA (swap pin 2 with pin 3) and WB pins (swap pin 4 with pin 5). For example, to reverse

A B C D, rewire as

B A D C.

For the purposes of testing, the default power-on rate of 100 ½-steps/second should work with most motors. Otherwise, use the serial connection to define the precise rate needed.

Single motor, double current mode of operation

All of the drivers on our boards can be operated in parallel, in order to double the current available. However, due to timing issues, our updates to each motor are usually offset by 8 microseconds; therefore, in prior firmware releases, you could not simply connect a single motor to the X and Y connectors in parallel, and expect correct operation of the system -- even if you always ran both motors identically. Actually, you would be shorting power to ground!

You can configure the board to send the same signal to the Y motor as is sent to the X motor (with the internal Y operations ignored). If you then wire your motor to BOTH the X and Y connectors (in exact parallel, so that (for example) WA1 from both the X and Y connectors is connected to your Winding A, pin 1 of your motor), then the board can provide double its normal per-winding capacity.

You configure the board to operate this way by connecting a shorting shunt in jumper location R1K.

You then wire your motor to BOTH the X and Y connectors (as described above); double the current will be available. Please note that if you do not correctly do the above wiring, then you will not get the benefit of the double power mode, and the board is quite likely to fail.

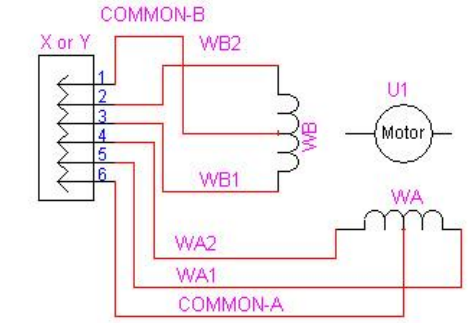
Motor Wiring Examples

The systems have been tested with an interesting mix of unipolar stepper motors. All were purchased from Jameco (www.jameco.com). The following sections summarize some of the motors tested. Note that BiPolar motors are NOT supported by the SS0805.

The wiring diagrams shown are labeled for the BiStepA05 and SimStepA04. The SS0805 wiring is identical.

Unipolar Motors

This section shows some unipolar motors which were used. Most will work on any of the boards currently available from our company. In each case, the wiring is:



Typical Unipolar Motor Connection
To the BiStepA05 or SimStepA04

Jameco 105873 12 Volt, 0.150 Amp/winding, 3.6 deg/step

This Howard Industries stepping motor has a manufacturing part number of 1-19-4202. It is wired as:

Color	SS0805
Black	1
Brown	2
Red	3
Green	4
White	5
<no connection>	6

Jameco 151861 5 Volt, 0.55 Amp/winding, 7.5 deg/step

This Airpax motor has a manufacturing part number of C42M048A04. As with the other Airpax motor, it does not microstep at all. Mode "3o" can smooth its actions, but it does not "stop" at any other points than ½ step locations. It is wired as:

Color	SS0805
Green	1
Black	2
Brown	3
Yellow	4
Orange	5
Red	6

When using a 5 volt motor (such as this), you may use a single, 6.5 volt power supply (this may slightly over-voltage the motor), or you may use a split supply. In this case, use a 6.5-12 volt supply for the power to the digital electronics (pins Vc and GND on the power connector), and a 5 volt power supply for the motor (pins Vm and GND on the power connector). **You will also have to run the motor using the 'Double Current' mode of the controller, as described elsewhere in this manual.**

Jameco 155432 12 Volt, 0.4 Amp/winding, 2000 g-cm, 1.8 deg/step

This motor provides for 2000 g-cm of holding torque, and has a manufacturing number of GBM 42BYG228. Its wiring order is:

Color	SS0805
White	1
Brown	2
Yellow	3
Red	4
Blue	5
Black	6

Jameco 162026 12 Volt, 0.6 Amp/winding, 6000 g-cm, 1.8 deg/step

This motor provides for 6000 g-cm(!) of holding torque, and has a manufacturing number of GBM 57BYGO84. Its wiring order is:

Color	SS0805
Black	1
Orange	2
Green	3
Yellow	4
Blue	5
White	6

Jameco 169201 24 Volt, 0.3 Amp/winding, 1.8 deg/step

This excellent motor has a manufacturing part number of STP-57D317. It uses 6 wires, with the wiring being:

Color	SS0805
Black (Common lead for PEACH and VIOLET)	1
Peach	2
Violet	3
Yellow	4
Red	5
White (Common lead for Yellow and White)	6

Jameco 173180 12 Volt, 0.060 Amp/winding, 0.09 deg/step geared

This tiny motor has a manufacturing part number of 30BYJ02AH, BF33. Thanks to its gearing, it claims to have both a holding and detent torque of 400 g-cm! It uses 5 wires, already in a connector which directly works with our product. However, two of the wires must be switched (i.e., the order of the wires is incorrect for our use): the pink and yellow wires need to be reversed in the connector. The correct order therefore becomes:

Color	SS0805
Red	1
Orange	2
Pink	3
Yellow	4
Blue	5
<no connection>	6

Jameco 174553 12 Volt, 0.6 Amp/winding, 7.5 deg/step

This motor has a manufacturing part number of NMB PM55L-048-NBC7. Its wiring is:

Color	SS0805
Black (common for Brown and Red)	1
Brown	2
Red	3
Green	4
Yellow	5
Orange (common for Yellow and Green)	6

This may only be operated with the SS0805 if you use the double current mode of operation!