SMC-400, 401, 401PS, 401PSA, 500, 501, 501PS, 501PSA, 600PS, 600PSA

SMC40x, SMC50x Series
Bipolar Chopper
Intelligent Stepping Motor Controllers
Fullstep, Halfstep, Microstep
(including integral power supply types)

SMC-600PS, SMC-600PSA
Unipolar Chopper
Intelligent Stepping Motor Controllers
Fullstep, Halfstep, Microstep
(integral power supply types)

Author: R. Schauer

Revision I, 1/11/97

Kollmorgen / Industrial Drives 201 E. Rock Road, Radford, VA 24141 (540) 633-4174
General Information

Kollmorgen SMC-40x, 50x, 60x series step motor controllers are designed to accept high-level commands via an onboard RS-232C-compatible multidrop port. Features include an efficient bipolar chopper step motor output stage, general purpose inputs, general purpose outputs, home and extension limit inputs, a "stop" input, and a busy output line. 24-bit position registers are provided.

Basic SMC-400 and 401 units have seven inputs (four general purpose inputs, the home and limit inputs, and the stop input) and five outputs (four general purpose outputs and the busy output). The inputs are buffered by optical isolators for protection and have 2700 ohm series resistors referenced to +5 volts. The outputs are open collector types capable of sinking up to 500 milliamperes each at up to 50 volts maximum. The SMC-400-40 and 401-40 options allow operation at up to 42 volts.

Basic SMC-500, 501, 501PS and 501PSA units have seven inputs (four general purpose inputs, the home and limit inputs, and the stop input) and five outputs (four general purpose outputs and the busy output). The inputs are buffered by HCMOS buffers with TTL pullup resistors. The outputs are open collector types. The SMC-500-40 and 501-40 options allow operation at up to 42 volts.

SMC-401PS, 401PSA, 600PS, and 600PSA units have eleven inputs (eight general purpose inputs, the home and limit inputs, and the stop input) and nine outputs (eight general purpose outputs and the busy output). The inputs are buffered by optical isolators for protection. The outputs are open collector types.

Motor Drive-

The output driver bridge is capable of driving up to 2 amperes (SMC-400/500), or 3.75 amperes (SMC-401/501) into each phase of a two-phase bipolar step motor. SMC-600 units are capable of driving up to 5.5 amperes into each phase of a four phase unipolar step motor. The motor winding current is limited by means of 25 kilohertz chopper logic. The onboard potentiometer is used to vary both winding currents and is marked 0-100 % on the cover. The nature of the chopping scheme eliminates the need for external current limiting resistors for the motor windings, simplifying connections and increasing the overall efficiency.

All SMC40x/50x/60x controllers have the capability of reducing the winding currents by approximately 50% at idle automatically. This is an internal ratio and is not changeable by the user.

Power Sources-

For basic units (no internal supply) the motor supply voltage should be at least 12.5 volts for logic operation, but must never exceed 35 volts. It is necessary for the motor supply to be filtered DC, but regulation is not normally required. A range of 16 to 32 volts is considered typical, with 24-30 volts preferred. Units capable of operation at voltages up to 45 VDC are available upon special request. It is the system integrator's responsibility to provide sufficient filtering of the incoming power to reduce power supply ripple to less than 1 volt under peak load conditions (see "Electrical Considerations" for more information). The SMC unit itself does not have sufficient power source filtering internally to handle the peak motor phase loads. Under no circumstances may you rectify the output of a transformer and supply it directly to these units without filter (smoothing) capacitors.

Units with integral power supplies (-PS and -PSA units) accept 100-130 or 200-260 volt AC power input (switch selectable) at any frequency from 48 to 63 hertz. All internal voltages are derived from this input. The internal motor supply voltage is available for external use (up to 2 amperes, unregulated 36-44 volts). On -PSA models, there is an additional regulated output of 23-26 volts provided (up to 1 ampere).

Internal voltages required for operation are derived from the stepper motor supply.
Software

The command set is very versatile and easy to use. Branches, loops, conditional tests, and more are provided. Both decimal and hexadecimal numeric modes are supported. There are seven 64-character non-volatile memory buffers provided along with a 64-character user buffer, subroutine support, and power up autoexecute functions. The "setup" parameters are also stored in non-volatile memory (EEPROM).

Stepping pulses can be input to the controller directly, if so desired, by means of the "Follow" command which allows two of the general purpose inputs to be used as step and direction. Normally however, the board generates all step pulses internally, in response to commands sent via the RS-232C port. For homing, jogging, logical seeks, relative moves, and auxiliary seeks, all ramping, slewing, and motor current logic is done automatically. The user need only set the minimum, maximum, acceleration rate, deceleration rate, and idle current options. In the case of -500 series units, it is also possible to separately specify the positioning mode (step resolution) and the slew-motion mode. These settings may be saved to the onboard non-volatile memory, and they will automatically loaded at power up.

SMC-40x/50x/60x controllers allow independent selection of the acceleration and deceleration ramp slopes, and also have a jog rate divisor which allows extremely slow jog speeds when desired. Slewing speeds up to 10,000 steps per second may be programmed for -40x/60x controllers, and up to 6450 steps per second for -50x controllers. The -50x controllers also have the ability to make motions at multiples of the selected resolution mode (up to 412,800 microsteps/second equivalent stepping speed).

-notes-

SMC-40x/50x/60x controllers allow independent selection of the acceleration and deceleration ramp slopes, and also have a jog rate divisor which allows extremely slow jog speeds when desired. Slewing speeds up to 10,000 steps per second may be programmed for -40x/60x controllers, and up to 6450 steps per second for -50x controllers. The -50x controllers also have the ability to make motions at multiples of the selected resolution mode (up to 412,800 microsteps/second equivalent stepping speed).

-notes-
SMC-40x/50x (non-PS/PSA) units-

SMC-40x/50x step motor controllers are supplied with baseplates which act as both mounting surfaces for the power devices on the board, and as a heatsink for them. NEVER operate the board when removed from the baseplate, as irreversible damage to the output driving devices will occur.

Under high ambient temperature, or heavy usage applications, it may also be necessary to mount the baseplate to a larger metal heatsinking surface through the use of the four 4-40 threaded mounting inserts in the bottom flange of the baseplate. It is suggested that the heatsink be at least 1/8" in thickness. The mass or area will depend upon the specific application, but in general should be sufficient to dissipate enough heat to maintain the SMC's baseplate at no higher than 50 degrees C (122 F) under operating conditions. Thermal dissipation characteristics of less than 4 degrees centigrade per watt are normally desired.

These units generally weigh about 12.6 ounces (360 grams), so mechanical support is rarely a concern except in high vibration environments.

Important -- mounting screws or bolts must not protrude more than 1/4" (0.25") in toward the board past the mounting surface.

---

SMC-401PS/4-1PSA/501PS/501PSA/600PS/600PSA units-
"-PS" and "-PSA" controllers have larger bases and enclosures, suitable for more rugged environments. These units are mounted by means of the slotted tabs at their bases.

Under high ambient temperature, or heavy usage applications, it may also be necessary to mount the base to a larger metal heatsinking surface. The mass or area will depend upon the specific application, but in general should be sufficient to dissipate enough heat to maintain the SMC's baseplate at no higher than 50 degrees C (122 F) under operating conditions, as measured at the base slotted tabs.

These units generally weigh around 6.5 pounds (2.95 kilograms) each, so stout mechanical mounting surfaces are a concern. It is recommended that 1/4" diameter fasteners and backup washers should be used to fasten the unit to the mounting surface.

Electrical Considerations-
Power Sources-
For non-PS/PSA units (no integral power supply), it is very important to provide low impedance clean and stable DC power to the units. It does not have to be regulated, but care should be taken that it does not exceed the voltage rating of the controller (up to 35 volts for regular units or up to 45 volts absolute maximum for the "-40" option). Even voltage spikes count, and can cause irreversible damage to the controller if these limits are exceeded. The power source must be capable of supplying the peak winding current that the motor requires. For example, if a motor requires 2 amperes per phase, the peak current requirement is 4 amperes (both phases on). The supply must be capable of providing this current without "sagging". A maximum voltage ripple of 1 volt as measured at the SMC's power input can be safely tolerated. Many drive problems (including overheating and instability) have been traced directly to inadequate power sources.

Power Supply Sizing (non-PS/PSA units)-
Because of the efficient current chopper drive logic, it is necessary to supply only a little more power than the motor will actually use. This seems only logical at first glance, but sizing a power supply can be confusing when examined closer. Let's use a practical example to show how it is done.

Let's assume that a 2 ampere / 2 ohm per phase motor is used. At 2 amperes current draw, a 2 ohm phase would require a drive voltage of 4 volts. Obviously, 16 watts of power is required for the two phases combined.... well sort of. At idle (pulse rate equals zero), this may be more or less true because the power input is being converted to pure holding torque. There is no motor "back-EMF" and few inductive effects to consider.

Assume also that we have a 24 volt power source (typical). The SMC controller will switch the incoming power on and off very rapidly (about 25,000 times per second) to regulate the motor winding currents by means of pulse width modulation. A very simplified rule-of-thumb calculation would show that the power should be turned on about 16.7% of the time (for 4 volts on the windings). In reality, other factors will alter this percentage, but it is a fairly close estimate. Since the power is turned on about 1/6th of the time, the average current supplied by the power source will be just 0.67 amperes (2 amps/phase x 2 phases x 0.167 on time). The wattage being supplied by the power supply is therefore about 16 watts. In reality though, it will be a bit higher if for no other reason than the losses in the switching circuitry, but again it is close enough for our purposes. Does this mean that you can use a 24 volt 0.67 amp power supply to drive this motor? No.

The first problem is that despite the average current draw being only 0.67 amps, the peak current draw is still 4 amps. Particularly if you are using switching power supplies that have high speed current limiting, the supply voltage ripple could be 20 volts or more due to current foldback! You can solve this by adding enough capacitance in parallel across the power wiring (preferably very close to the SMC unit) to handle the peak currents and help to smooth out the average demand upon the power source. You may also need to add some inductance between the capacitor(s) and the output of the power supply (never, never add it between the capacitors and the SMC unit!).

Let's say we find a capacitor that will handle 2 amps ripple current at 20 kilohertz (more or less the chopping frequency). Typical examples would include Sprague “672D/673D”, UCC “SXC”, Rubycon “G2A”, Nichicon “PX”, and Panasonic “HF/HFU” series capacitors in the 390 to 560 microfarad range. If we parallel two of them, the heavy current peaks will be supplied by the capacitors, not the power supply. It is better to parallel smaller capacitors to obtain the same ripple capacity than to select one huge one due to the internal impedances (ESR) of the capacitors. Use "high frequency" or "photoflash" capacitors whenever possible. The capacitance provided on the SMC boards is merely to handle high speed transients and to attempt to nullify some of the effects of the power wiring resistance and inductance. It will not boost up an inadequate power source nor will it handle the entire motor current load alone.

The next problem is that the power requirement mentioned above is for a motor at rest. What happens when it actually does work (movement)? Stepping motors are like servos and other types of permanent magnet motors so far as their behavior when in motion. The mere fact that they are in motion will cause them to generate voltage (“back EMF”). The generated voltage is the same polarity as the driving voltage, so it bucks the supply voltage, thus reducing the amount of voltage available to generate torque (the "overhead" voltage). When the motor is moving sufficiently fast, its generated voltage can
meet (or even exceed) the supply voltage. What happens in these situations is that there is literally no voltage left to drive the windings and generate torque. When this happens, the motor’s output torque is reduced (possibly to zero), and the motor will probably “stall”. Solution? Use more supply voltage or a lower voltage motor. That is why it is better to drive the motor with a voltage much higher than its \( I/R=E \) "nameplate" rating. To do otherwise would result in very poor motor performance indeed.

So what does this have to do with the current output of the power source? Let us assume that the generated back EMF of the motor is 16 volts at a certain speed (pulse rate). If so there is only 8 volts available to generate torque. Since the motor still requires 4 volts to generate full torque, and the SMC control will do its best to regulate the current to that level, at this point the chopper circuitry will be on at least 50% of the time. This means that the 4 ampere peak current requirement will be "on" 50% of the time. Now you need an average of 2 amperes from the supply at least. This simplistic view ignores inductance effects, chopper logic switching time, diode and transistor losses and other effects, but gets the point across (those things may actually make the situation worse). Where does all this extra wattage go? It goes primarily into motor output power (torque at a rotational speed) and to a lesser extent into motor and driver heating. Now at least 48 watts of power are required to generate the full torque at speed (up from only 16 watts at rest). This only makes sense given that "work" is being performed by the motor. The power for that work must come from somewhere.

The average power supply requirement should now be at least 2 amperes at 24 volts. If the motor is to be run at or near the peak of its power/speed range, you would probably need the full 4 amperes at 24 volts (and remember this is for only a 2 amp per phase motor!). In general, a rule of thumb is to select a power supply capable of supplying 125% of the peak power that is required.

No matter which supply is chosen, pay very close attention to the filter caps provided by the power supply manufacturer. Ripple current capacity must be considered if stable, reliable operation is to be obtained. Please note that on -PS or -PSA units, this has all been taken care of for you on the internal power supply. Although the “RMS” current capacity of the internal supplies are only a little greater than the peak current requirements of the drive, the ripple capacity of the internal filter capacitor bank is considerably higher. This promotes stable and reliable operation.

**Wiring and Ground Loops**

Another potential problem area that is often overlooked is the wiring between the power source and the SMC control. Situations where long power wiring runs or small wires are used are just asking for trouble. The power source voltage ripple as seen at the SMC unit will be greatly magnified in such situations.

As an example, if 10 feet of 20 gauge stranded wire is used as power input wiring with no chassis ground at the SMC end (making the total wiring run 20 feet including the ground return), the total wiring resistance would be 0.2 ohms. If no "stiffening" capacitor is placed at the SMC's power input, there could easily be 0.8 volts of ripple due to I/R effects alone (0.01 ohms per foot times 4 amperes peak for the 2 amp per phase motor in the example), not to mention several volts of spikes due to wiring inductance effects, plus whatever ripple the power supply itself allows. These problems can cause major headaches, drive instability, and can even result in driver damage.

Power wiring should be kept short and small in gauge number (large conductor area). If the motor is to be located some distance from the power source, run the long wires in the motor side of the circuit, not in the power side. The inductances and resistances of the motor-side wiring will probably be "lost" in the inductance and resistance of the motor itself anyway. On the other hand, it is very important to have the SMC driver coupled closely and tightly to the power source.
Remember also that any voltages appearing on the ground side of the SMC (for any reason) are reflected directly to the ground pins of the inputs, outputs, and RS-232 serial connectors. High frequency common-mode voltage spikes can easily cause problems with standard serial ports on PC's and other communications devices. Always make sure that the ground side of the circuit wiring has very, very low impedance. Use the framework of the machinery into which the SMC is mounted as a ground plane wherever possible. Avoid ground loop potential voltages!

Summary-
When sizing the motors and drives, you need to examine the motor's speed/torque/power curves to determine the peak power requirements. It is not sufficient to simply look at the nameplate rating of the motor and choose a supply. As mentioned in the previous discussions, the maximum power requirements must be determined and a safety factor added (typically about 25%). Please consult your Kollmorgen representative for details concerning specific motor and drive combinations.

It is entirely possible to mechanically overdrive stepping motors just as can be done with servos. In fact, higher back EMF voltages than the motor supply itself can easily be generated. If this occurs and nothing is done to clamp the voltage, it is possible to damage the driver output bridge. The "regeneration" resistors in large servo drives are there partly to prevent this sort of occurrence. There is no physical space inside an SMC drive for them, and they can be rather costly to apply as well.

Be careful in your mechanical designs to prevent or limit unintentional motor backdrive or overdrive conditions. Alternatively, provide a method to limit the maximum supply voltage under adverse conditions if they may possibly occur. A power zener diode or a shunt regulator circuit across the supply voltage rail and ground work very well for limiting occasional transients. Just be sure to select a clamping voltage comfortably above the supply's peak output voltage but within the safe limits of the SMC controller.

-notes-
**Connections / Pinouts**

**ALL Units**

**J2- Serial Input/Output**
Mating Plug Style- DB-9-Male, AMP 747904-2 (solder cup style) or functional equivalent.
- J2-2 Transmitted serial data output
- J2-3 Received serial data input
- J2-5 Board ground/Serial Common

**J3- Motion Limits, Status**
Mating Plug Style- MOLEX 22-01-3067 or equivalent.
- J3-1 Board ground
- J3-2 < Home Limit> input
- J3-3 <Extension Limit> input
- J3-4 Hardware Stop input (Capture)
- J3-5 <Busy> output
- J3-6 Board ground

**J4- Motor Power Input/Output (where equipped)**
Mating Plug Style- MOLEX 09-50-3031 or equivalent.
- J4-1 Board power ground
- J4-2 Motor power input (non-PS units) or internal voltage bus output (-PS, -PSA units)
- J4-3 Board power ground

**J6- Auxiliary Power Output ( -PSA versions)**
Mating Plug Style- 1.3mm x 3.4mm x 9.1mm DC power plug (any vendor)
- J6-center pin +23-35 volts regulated output
- J6-shell Board power ground

**SMC-400,401,500,501,501PS,501PSA units ONLY**

**J1- General Purpose Inputs/Outputs**
Mating Plug Style- MOLEX 22-01-3107 or equivalent.
- J1-1 Board ground
- J1-2 General purpose input 0
- J1-3 General purpose input 1
- J1-4 General purpose input 2
- J1-5 General purpose input 3
- J1-6 General purpose output 0
- J1-7 General purpose output 1
- J1-8 General purpose output 2
- J1-9 General purpose output 3 (up to 750 ma., this output ONLY)
- J1-10 Board ground

**J5- Motor Connections**
Mating Plug Style- MOLEX 09-50-3041 or equivalent.
- J5-1 Motor phase A start of phase winding
- J5-2 Motor phase A end of phase winding
- J5-3 Motor phase B start of phase winding
- J5-4 Motor phase B end of phase winding

**SMC-401PS,401PSA,600PS,600PSA units ONLY**
**J1- General Purpose Inputs/Outputs**
Mating Plug Style- AMP 1-87456-6 (with contact pins) or equivalent for individual wires, OR AMP 746285-4 or equivalent for ribbon cabling.

- J1-1,2 Board grounds
- J1-3 General purpose input 4
- J1-5 General purpose input 5
- J1-7 General purpose input 6
- J1-9 General purpose input 7
- J1-4 General purpose input 0
- J1-6 General purpose input 1
- J1-8 General purpose input 2
- J1-10 General purpose input 3
- J1-11 General purpose output 4
- J1-13 General purpose output 5
- J1-15 General purpose output 6
- J1-17 General purpose output 7
- J1-12 General purpose output 0
- J1-14 General purpose output 1
- J1-16 General purpose output 2
- J1-18 General purpose output 3
- J1-19,20 Board grounds

**J5- Motor Connections**
Mating Plug Style- MOLEX 22-01-3067 or equivalent.

- J5-1 Motor phase A start of phase winding
- J5-2 Motor phase A end of phase winding
- J5-3 Motor phase A common *
- J5-4 Motor phase B common *
- J5-4 Motor phase B start of phase winding
- J5-4 Motor phase B end of phase winding

*the two commons may be connected together without harm

The locations of the connectors are as shown-

![Diagram](image-url)

The connectors are accessible from the sides (ends) of the units. Suitable cutouts or accesses have been provided. The connectors generally have some form of detents or fasteners incorporated into them to prevent unwanted disconnection due to vibration or stress.

**Inputs and Outputs**
Inputs-

The general purpose input lines are TTL/CMOS-compatible, and can be sampled from the program string (your commands), or can be used as conditional inputs for jog, follow, home, or conditional loop modes. They are all optically isolated and have 2700 ohm series resistors referenced to +5 volts internally. Sinking an input to less than about 1.3 volts turns it on. When "off", the program statements will sense it as a logical high ("1"). When "on" it will be a logical low ("0"). There are four general purpose inputs on non-"PS" units, and eight on "-PS" and "-PSA" (except 50xPS) units. They are numbered 0-3 or 0-7 as applicable.

A logical low on the HOME input (J3, pin 2) stops any counterclockwise (negative direction) motions in progress. The motion is aborted, and will not resume if the input low is released unless specifically commanded to do so (another move, seek, or home command). Similarly, a logical low on the LIMIT input (J3, pin 3) stops any clockwise (positive direction) motion in progress.

The STOP (labeled either CAPTURE or STOP) input (J3, pin 4) can be programmed to be either positive or negative edge sensitive. When activated, any motion or program command in progress will be stopped. Either of them will not resume until specifically commanded to do so. When this input is used, the "autoexecute" flag is not affected. In conjunction with the programmable restart function, the controller may be both started and stopped with simple inputs (without RS-232C being used).

Outputs-

The general purpose outputs can be switched from the program string (by your commands), and are open-collector types capable of sinking 500 ma. each, except for OUT3 on non-"PS" units, which can sink 750 ma. Supressor diodes referenced to the internal motor supply voltage are provided for all outputs. The "snoopy" command can also drive the general purpose outputs. When turned on, the outputs will sink their rated current to less than 0.5 volts. There are four general purpose outputs on non-"PS" units, and eight on "-PS" and "-PSA" (except 50xPS) units. They are numbered 0-3 or 0-7 as applicable.

The BUSY output (50 ma. max) will be on (low) whenever the motor is in motion. This line is functional during all types of motions. There is no suppressor diode on this output, being intended for connection to external logic as opposed to inductive loads.

Serial I/O-

The TXD data line (J2, pin 2) is only active during transmissions of data to the host device from the motor controller board. It can be switch-selected to be multidrop compatible. When the non-multidrop mode is selected (MULTI switch off, TERM switch on), this output will be driven low at idle (-3 volts or less), and will be driven high only during transmissions (+3 volts or more). When in the multidrop mode (MULTI switch on), it is not driven low at idle unless the TERM switch is turned on. One and only one SMC controller in the "multidrop" chain should have its TERM switch on.

The RXD data line (J2, pin 3) always listens to the host's data outputs. If switch-selected as a multidrop device, the motor controller will only respond to messages containing its particular address. Please refer to the multidrop communications section for further details.

Baud rates up to 9600 are supported. Eight bits of data, no parity, one start bit, and one stop bit are standard. If your host device must send a parity bit, select 7 data bits for its interface (plus the parity). This totals eight bits, and the SMC's will ignore the eighth (parity) bit in any case.

The sending of an "ESC" character (1B hex) will stop any action or motion in progress. If the multidrop mode is selected, it will have to be preceded by the normal multidrop preamble, which is: <STX>-ADDRESS (see the multidrop communications section for further details).
Inputs (SMC400, 401, 401PS, 401PSA, 600PS, 600PSA)-

Inputs (SMC500, 501, 501PS, 501PSA only)-

Outputs (all units)-

Note- The BUSY output does not have the suppressor diode.
Differences Between 40x and 50x Units-

The SMC-400/401 series (including -PS and -PSA) differs from the SMC-500/501 series mainly in that the -40x series is capable of full, half, and "wave" drive modes only, whereas the -50x series is capable of up to sixty-fourth step resolution modes. The -40x series drives are full-phase-on type drives, relying upon sequence changes to achieve the various resolution modes. The -50x series on the other hand, uses synthesized sine and cosine waves of varying degrees of resolution to achieve its modes. The SMC-40x series will generally be capable slightly higher torque generation (for equivalent drive modes, winding currents, supply voltages and motors) because of the full-phase-on methods. Please consult your Kollmorgen representative for proper selection of drive and motor combinations.

SMC-40x Stepping Modes-

As mentioned, the SMC-400/401 and derivatives have three basic stepping modes. They are as follows-

The **fullstep** mode sequences the motor phases in the following manner-

<table>
<thead>
<tr>
<th>Phase A</th>
<th>Phase B</th>
<th>Step #</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>+</td>
<td>0</td>
</tr>
<tr>
<td>+</td>
<td>+</td>
<td>1</td>
</tr>
<tr>
<td>+</td>
<td>-</td>
<td>2</td>
</tr>
<tr>
<td>-</td>
<td>-</td>
<td>3</td>
</tr>
</tbody>
</table>

The full step mode provides the maximum low speed torque because two windings are always energized. It also provides the largest amount of rotation per step pulse. Since the fullstep mode moves the farthest per pulse, fewer pulses can be used for a given motion, hence a slower pulse rate is used for any given rotation rate. Unfortunately, fullstep modes are also the most prone to motor and drivetrain resonances (often leading to unstable motor "speed" regions). Careful mechanical and software design practices are normally required to effectively use the fullstep mode.

The fullstep mode is also the most "power-hungry" on the average, because both phases are always energized. This in turn means that the output driver and motor heating will be most apparent.

The fullstep mode will almost always be the noisiest acoustically, and has the highest mechanical torque ripple. Conversely, the electrical current ripple is lower than the halfstep mode, which alternates between one and two phases being energized.

The **halfstep** mode sequences the motor phases in the following manner-

<table>
<thead>
<tr>
<th>Phase A</th>
<th>Phase B</th>
<th>Step #</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>+</td>
<td>0</td>
</tr>
<tr>
<td>off</td>
<td>+</td>
<td>1</td>
</tr>
<tr>
<td>+</td>
<td>+</td>
<td>2</td>
</tr>
<tr>
<td>+</td>
<td>off</td>
<td>3</td>
</tr>
<tr>
<td>+</td>
<td>-</td>
<td>4</td>
</tr>
<tr>
<td>off</td>
<td>-</td>
<td>5</td>
</tr>
<tr>
<td>-</td>
<td>-</td>
<td>6</td>
</tr>
<tr>
<td>-</td>
<td>off</td>
<td>7</td>
</tr>
</tbody>
</table>
The halfstep mode usually provides the smoothest mode of operation. It also provides the smallest amount of rotation per step pulse available with the SMC-40x drivers. Its principle advantage is a much higher resistance to mechanical motor and system resonance. For this reason, higher usable motor angular rotation speeds are usually possible with the halfstep mode, although this may not seem logical at first glance.

The -40x halfstep mode alternates between one and two phases on, and therefore exhibits the highest supply current ripple of the three available drive modes. This is not normally a problem, but careful attention may have to be paid to supply filtering and decoupling between motor drivers if many motors are to be driven from the same supply (even if it is regulated). The halfstep mode usually produces the acoustically quietest operation, and has less average torque ripple than the fullstep mode, although the torque peaks are uneven.

The halfstep mode, while it is actively stepping, is not as power hungry as the fullstep mode. Less output driver heating is a side benefit. If the motor is stopped with two phases on however, there is then no difference between the modes. The halfstep mode allows higher stepping rates than either fullstep or wave drive. This is because the motor is being "pulled" magnetically through smaller increments when rotating in the halfstep mode. For this reason, the maximum net (usable) rotation speed may actually be higher than in the fullstep mode (although of course at twice the step pulse rate).

The wavedrive (or "one phase on") mode is a variation on the fullstep mode which exhibits the following phase pattern:

<table>
<thead>
<tr>
<th>Phase A</th>
<th>Phase B</th>
<th>Step #</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>off</td>
<td>0</td>
</tr>
<tr>
<td>+</td>
<td>off</td>
<td>1</td>
</tr>
<tr>
<td>off</td>
<td>-</td>
<td>2</td>
</tr>
<tr>
<td>off</td>
<td>+</td>
<td>3</td>
</tr>
</tbody>
</table>

The wavedrive mode provides the lowest power consumption of any of the three -40x modes. One phase is always on, but never more than one. The step angle per step pulse is the same as in the fullstep mode, but with less low-speed torque available.

There is an important advantage to this mode concerning step angle accuracy. At times in the 40x halfstep or fullstep sequences, the rotor position is dependent upon the balance between the magnetic fields of adjacent windings (whenever two phases are on simultaneously). Because the fields are dependent upon so many factors, they are much more difficult to control than machining accuracies. When a single phase is enabled, the rotor aligns with the magnetic field of the single phase pole, which is fairly accurate mechanically. Since the wavedrive mode uses only single phases in its sequence, the step-to-step accuracy is normally superior to either of the other modes.

Naturally, since only one phase is ever on in this mode, the power consumption is the lowest, whether running or stopped. The supply current ripple is lower than the halfstep mode, and the torque ripple is low (the steps being "softer"). Output driver heating is also minimal in this mode.

There is of course, a price to pay for the advantages of this mode. One is that it has the least average available output torque of the three modes. A second is that it has the same step angle per pulse as the fullstep mode. A third is that it is both acoustically noisier and more prone to resonances than the halfstep mode (for much the same reasons as in the fullstep mode). It is however, the mode of choice for battery operated equipment and any other applications where low supply current or heating are paramount. It is neither a high speed, nor a high torque mode, but rather a "low power" operating mode.
The SMC-500 and -501 series of drives are capable of splitting each mechanical "step" into as many as 64 subparts. The stepping method is the same in all modes: synthesized sine and cosine waves of varying resolutions. This means (in the instance of a standard 1.8 degree step motor) theoretical resolutions as small as 0.028125 degrees per step. In practice, very high precision, low detent torque motors are required to even get close to such accuracy, but the -50x series is capable of generating the appropriate drive waveforms nonetheless. Normally the ultra-fine resolution modes show their greatest usefulness in solving slow-speed, high smoothness, or high-accuracy motion problems, and in the virtual elimination of motor and system resonances.

The SMC-500/501 microstep controllers have the ability to drive extremely rapid rotational motions, even while set to high-resolution positioning modes. This is achieved by the use of mode indexing, which is basically the ability to switch drive modes on the fly, thus performing the bulk of a commanded motion in a faster (lower resolution) mode than the one selected for positional accuracy. When this feature is used, its operation is completely transparent to the user. The user need only command the desired positioning mode, the desired slewing (gross motion) mode, and the number of steps to go in the positioning mode. The rest is automatic.

Note that while all this is occurring, the absolute position counter is updated in the positioning mode's counts. That is to say that if you have selected the 1/64 step mode for positioning, and the 1/2 step mode for slewing, during any subsequent motions the position counts will be kept in 1/64th step increments. Therefore, if you command 10,000 steps to be done, the equivalent of 10,000 1/64th steps will be done, and the position counter will reflect 10,000 steps as having been done (even though the bulk of the motion may actually have been done in 1/2 steps). If you have selected 1/32 steps for the positioning mode, counts will be kept in 1/32nd steps, etcetera.

While this is a departure from traditional step motor drive practice, and may in some instances require some experimentation to suit your particular application, the benefits should be obvious. You can position to whatever accuracy you desire (up to 1/64 of "full" steps), but still get from position-to-position rapidly. Motion times and accelerations should be calculated using the "slewing mode" resolution. Once set up, these actions are completely transparent to the operating program and user.

Should you not want this feature, simply set the positioning mode equal to the slewing mode (see the "Dn" and "Dxn" commands) and motions will be made in the positioning resolution selected without mode indexing.

*Whenever the slewing and positioning modes are different, the auto-idle-power-down mode is automatically turned off.* This is because the motor needs a full-torque starting position to begin any given motion. As per the discussion of the SMC-40x's "wave" drive mode, the motor will tend to align itself with the nearest magnetic pole positions when the winding currents are reduced or shut off. This self-alignment tendency is acceptable in a full-phase-on drive system (-400/401), but can be highly detrimental in a sine/cosine system.

The difference between drive resolution modes is simply a difference in the resolution of the sine and cosine waves being generated. In the full-step mode for example, the phases are driven in a full-phase-on sequence similar (but not identical) to the SMC-400/401 "wave drive" mode. In the half-step mode, there are states in which the phases are approximately 70% on (sine of 45 degrees). As the step resolution increases toward the maximum of 1/64 step, the phase drive current waveforms increasingly conform to true sine/cosine curves. This is in fact the most natural mode of operation for the motor.

Since the output drives are current control pulse width modulation types (PWM), the voltage waveforms at the motor connections may not resemble sine waves very closely (if at all), but the current waveforms for the windings will show characteristic sine/cosine curves. Should you wish to examine them, use an isolated current probe or current transformer to interface to an oscilloscope or meter. NEVER, NEVER ground any of the motor output terminals, connect any of them to the power supply rails, or to any other inputs or outputs. To do so will damage the SMC unit, and will not be covered under warranty.

Microstep drive modes help to eliminate resonances in the drive system because, like the -40x "halfstep" modes, the motor is required to move less of an angle with each commanded step. As a result,
there is much less overshoot, less noise, faster damping, and improved low speed torque. Drive systems will experience less shock, thus prolonging their service life.

**SMC-60x Stepping Modes**

The SMC-600 and derivatives have three basic stepping modes. They are similar in nature to the SMC-40x series in function and characteristics and are as follows:

The **fullstep** mode sequences the motor phases in the following manner:

<table>
<thead>
<tr>
<th>Phase A</th>
<th>Phase B</th>
<th>Phase C</th>
<th>Phase D</th>
<th>Step #</th>
</tr>
</thead>
<tbody>
<tr>
<td>on</td>
<td>off</td>
<td>off</td>
<td>on</td>
<td>0</td>
</tr>
<tr>
<td>off</td>
<td>on</td>
<td>off</td>
<td>on</td>
<td>1</td>
</tr>
<tr>
<td>off</td>
<td>on</td>
<td>on</td>
<td>off</td>
<td>2</td>
</tr>
<tr>
<td>on</td>
<td>off</td>
<td>on</td>
<td>off</td>
<td>3</td>
</tr>
</tbody>
</table>

The **halfstep** mode sequences the motor phases in the following manner:

<table>
<thead>
<tr>
<th>Phase A</th>
<th>Phase B</th>
<th>Phase C</th>
<th>Phase D</th>
<th>Step #</th>
</tr>
</thead>
<tbody>
<tr>
<td>on</td>
<td>off</td>
<td>off</td>
<td>on</td>
<td>0</td>
</tr>
<tr>
<td>off</td>
<td>off</td>
<td>off</td>
<td>on</td>
<td>1</td>
</tr>
<tr>
<td>off</td>
<td>on</td>
<td>off</td>
<td>on</td>
<td>2</td>
</tr>
<tr>
<td>off</td>
<td>on</td>
<td>off</td>
<td>off</td>
<td>3</td>
</tr>
<tr>
<td>off</td>
<td>on</td>
<td>on</td>
<td>off</td>
<td>4</td>
</tr>
<tr>
<td>on</td>
<td>off</td>
<td>on</td>
<td>off</td>
<td>5</td>
</tr>
<tr>
<td>on</td>
<td>off</td>
<td>off</td>
<td>off</td>
<td>6</td>
</tr>
</tbody>
</table>

The **wavedrive** (or "one phase on") mode is a variation on the fullstep mode which exhibits the following phase pattern:

<table>
<thead>
<tr>
<th>Phase A</th>
<th>Phase B</th>
<th>Phase C</th>
<th>Phase D</th>
<th>Step #</th>
</tr>
</thead>
<tbody>
<tr>
<td>on</td>
<td>off</td>
<td>off</td>
<td>off</td>
<td>0</td>
</tr>
<tr>
<td>off</td>
<td>on</td>
<td>off</td>
<td>off</td>
<td>1</td>
</tr>
<tr>
<td>off</td>
<td>off</td>
<td>on</td>
<td>off</td>
<td>2</td>
</tr>
<tr>
<td>off</td>
<td>off</td>
<td>off</td>
<td>on</td>
<td>3</td>
</tr>
</tbody>
</table>

**Drive Methods, Notes**

If you need high performance, you should use low inductance, high current motors. They would be impractical in most R/L limited drives (due to current limit resistor heating), but are well suited to pulse width modulated chopper drives. These types of motors, even if not driven to their full nameplate current, will give very high running speeds with good torque throughout their operating ranges. Of course you can benefit from the idle current reduction to save power, too.

The SMC-40x/50x drives operate better at higher (but under 45 volts) motor supply voltages. The preferred input voltage is 28-30 (but must NEVER exceed 35 except on "+40" special order units). "-PS" and "-PSA" units have internal 40 volt (nominal) power supplies. Since the motor current is limited by varying the "on time" of the output drivers, the less time they are on, the lower their power dissipation due to "VCESAT" losses. The current sensing resistors also are "on" a lesser amount of time, which reduces their heating. From the board's viewpoint, therefore, the 28-40 volt range is ideal. Care must be taken so that even considering noise and spikes, the voltage input never exceeds 46 volts.
The motor winding current is easily set by the use of the potentiometer labeled "winding current" toward the lower left of the controller. The potentiometer is calibrated as 0% through 100%. This represents 0-100% of the output current rating of the drive (2, 3.75, or 5.5 amps per phase). The approximate winding currents (in amperes per phase) is as follows:

<table>
<thead>
<tr>
<th>Nameplate Setting</th>
<th>SMC400/500</th>
<th>SMC401/501</th>
<th>SMC600</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>10</td>
<td>0.20</td>
<td>0.38</td>
<td>0.55</td>
</tr>
<tr>
<td>20</td>
<td>0.40</td>
<td>0.75</td>
<td>1.10</td>
</tr>
<tr>
<td>30</td>
<td>0.60</td>
<td>1.13</td>
<td>1.65</td>
</tr>
<tr>
<td>40</td>
<td>0.80</td>
<td>1.50</td>
<td>2.20</td>
</tr>
<tr>
<td>50</td>
<td>1.00</td>
<td>1.88</td>
<td>2.75</td>
</tr>
<tr>
<td>60</td>
<td>1.20</td>
<td>2.26</td>
<td>3.30</td>
</tr>
<tr>
<td>70</td>
<td>1.40</td>
<td>2.63</td>
<td>3.85</td>
</tr>
<tr>
<td>80</td>
<td>1.60</td>
<td>3.00</td>
<td>4.40</td>
</tr>
<tr>
<td>90</td>
<td>1.80</td>
<td>3.38</td>
<td>4.95</td>
</tr>
<tr>
<td>100</td>
<td>2.00</td>
<td>3.75</td>
<td>5.50</td>
</tr>
</tbody>
</table>

Key considerations in the setting of the motor current include the nameplate rating of the motor used, the duty cycle for the motor, the use (or not) of the idle power reduction mode, and the torque required of the motor. In practice though, the motors themselves are not linear enough to respond exactly to changes in current, particularly at very low currents. In other words, setting the control to 10% will not necessarily yield exactly 1/2 the average current of a 20% setting. Please refer to your Kollmorgen representative for recommended setting for specific motor and drive combinations.

There is also the duty cycle to consider. For example, let us assume that you have a motor with a rated current of 1.5 amps per phase, and that you are requiring it to be actually stepping 80% of the time. Since the duty cycle is high, you should not exceed the motor's rated current. You may therefore adjust the current setting potentiometer to as high as 65-70% on an SMC-400 or 500 unit, or 35-40% on an SMC-401 or 501 unit or 25-30% for an SMC-600 unit (note that the base of the controller may require heatsinking). The motor should not overheat.

Let us assume that you are only requiring a 20% duty cycle with the same motor. You can actually overdrive the motor during motions if so desired, and turn on the automatic idle power reduction mode (preferably turn it off between motions). You could possibly set the motor current setting as high as 100% on an SMC-400 or 500 or 55-60% on an SMC-401 or 501 or 30-35% on an SMC-600 because at idle the current is reduced to 50% of whatever you set it at (or zero), and is not at the full current level much of the time. When you want motions performed, the motor's torque output will be increased over its normal capability, but the average winding current over time will be within the limits of the motor. Use caution when doing this because the motor may be destroyed by overheating should the duty cycle increase for any reason!

Overdriving is rather like heavily turbocharging an auto engine. It's acceptable for short term use, but you usually can't sustain it without damaging something. Note that on SMC-50x controllers, the reduction mode is automatically switched off if you have selected different positioning and slewing drive modes. Use extreme caution when overdriving. You should also be aware that past a certain point, the motor's laminations become magnetically saturated (they cannot be magnetized to a greater extent), so higher current merely results in heating of the motor windings and driver bridge components. No greater usable torque results in these situations, only a lot of heat. Motor burnout and damage to the controller is a likely outcome of this practice.
Motor Connections-

The motors used with the SMC40x/50x controllers need not necessarily be 4 wire types. Six and eight wire motors can be used so long as the two basic phases are electrically isolated from one another. The following diagrams show how the various types of motors may be connected.

The motors used with the SMC60x controllers must be 5 or 6 wire unipolar types. The following diagram shows how these types of motors may be connected. Note that the two phase commons (J5-3, J5-4) are connected together internally on the controller circuit card.
Drive Stability-

In certain circumstances (usually with lower supply voltages), the motor current regulation may be unstable at motor rest. This will be evidenced as a hissing or ringing noise from the motor. This is often due to the motor's natural time constant conflicting with the chopper clock, or from its lamination stack saturating magnetically. Another typical cause is inadequate power source wiring or excessive power source ripple for one reason or another. Excessive length, inductance or resistance in the motor wiring is also a cause of this problem.

The result is that some chop triggers are randomly missed. This in turn results in unstable overall chopping action and audible noise. This type of operation is undesirable as it will lead to increased heating of the output driver devices, and actually results in less net drive current than if the motor current setting is reduced slightly to achieve smooth chopping (no hissing). Stiffening (boost) capacitors may be required in the DC incoming power lines to help absorb the peak current loads. This sort of problem is usually not as apparent when using the -PS or -PSA drives, since they have adequate internal power filtering installed.

Please note that with bipolar chopping drivers (any manufacturer's) it is important to use reasonably low inductance windings. Your particular ideal current setting will vary with the motor and voltage you use. Please consult your Kollmorgen representative for proper motor/controller matches and recommended current level settings.

-notes-

19
Prior to all motions, the CPU checks for setup errors such as the maximum step rate being setup less than the minimum rate. In such cases, the maximum rate is adjusted to the minimum rate, resulting in a "flat" velocity profile.

The accelerations, decelerations, maximum and minimum rates, and other values, are set by using the "V" command. The syntax for this command is:

\[ Vnmm \]

Where \( n \) = register number
\( mm \) = value to be stored

Allowable values for registers:

<table>
<thead>
<tr>
<th>Number</th>
<th>Register Function</th>
<th>Range (hex)</th>
<th>Range (dec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Minimum Step Rate</td>
<td>0-7F</td>
<td>0-127</td>
</tr>
<tr>
<td>1</td>
<td>Maximum Step Rate</td>
<td>0-7F</td>
<td>0-127</td>
</tr>
<tr>
<td>2</td>
<td>Acceleration Slope</td>
<td>1-1F</td>
<td>1-31</td>
</tr>
<tr>
<td>3</td>
<td>Deceleration Slope</td>
<td>1-1F</td>
<td>1-31</td>
</tr>
<tr>
<td>4</td>
<td>Jog Rate Divisor</td>
<td>1-3F</td>
<td>1-63</td>
</tr>
<tr>
<td>5</td>
<td>Acknowledgement Byte</td>
<td>0-FF</td>
<td>0-255</td>
</tr>
<tr>
<td>6</td>
<td>General Status Byte **</td>
<td>0-FF</td>
<td>0-255</td>
</tr>
<tr>
<td>7</td>
<td>Input or Output Port</td>
<td>0-FF</td>
<td>0-255</td>
</tr>
<tr>
<td>8</td>
<td>Positioning Mode</td>
<td>0-6</td>
<td>0-6</td>
</tr>
<tr>
<td>9</td>
<td>Slew Mode (50x only)</td>
<td>0-6</td>
<td>0-6</td>
</tr>
</tbody>
</table>

( ** read only )

Minimum and Maximum Rates-
(registers 0 and 1)

<table>
<thead>
<tr>
<th>Register Value (hex)</th>
<th>Register Value (decimal)</th>
<th>Rate (SMC-40x/60x)</th>
<th>Rate (SMC-50x)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>16</td>
<td>50</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>94</td>
<td>100</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>173</td>
<td>150</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>251</td>
<td>200</td>
</tr>
<tr>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>7C</td>
<td>124</td>
<td>9764</td>
<td>6300</td>
</tr>
<tr>
<td>7D</td>
<td>125</td>
<td>9842</td>
<td>6350</td>
</tr>
<tr>
<td>7E</td>
<td>126</td>
<td>9921</td>
<td>6400</td>
</tr>
<tr>
<td>7F</td>
<td>127</td>
<td>10000</td>
<td>6450</td>
</tr>
</tbody>
</table>

For the SMC-400x/60x, each rate increment represents approximately 78.5 steps per second.
For the SMC-500,501, each rate increment represents approximately 50.0 steps per second.

Acceleration and Deceleration Slopes-
(registers 2 and 3)
The accelerations and decelerations are measures in steps per second per step. As can be seen above, the acceleration tapers off quite dramatically for increasing values placed in the acceleration and deceleration registers. Note that the numbers in the registers are inversely proportional to the acceleration and deceleration rate settings, which actually represent the number of steps done at each speed during up and down velocity ramping. Acceleration and deceleration settings much above 8 are allowed, but are of very limited use.

**Jog Rate Divisor-**
(register 4)

In the jog mode (Jnm command) or when the motion divisor is enabled, the stepping rate and accelerations are divided by the value contained in register 4. You can set this value by the command:

\[ \text{V4nn} \]
Where nn= value to be entered.

If you have the minimum and maximum rates set at say, 120 and 1200 steps per second, and you enter a value of 20 decimal (14 hex) to register 4, the resultant jogging step rate would be from 6 to 60 steps per second.

**Programmable ACK Byte-**
(register 5)

The acknowledge byte can be set to anything you wish, but the standard value is 06 hex/dec (ASCII "ACK"). To set this value, use the following command:

\[ \text{V5nn} \]
Where nn= the new "ACK" byte value.

It is important to choose a value for the acknowledge byte that will work harmoniously with the device you are using as a host controller. For example, a value of 7F hex would not be particularly advisable for most personal computers, as this is an ascii DEL character. You must choose what is appropriate for your particular application.

**Motor Microstepping Resolutions (SMC50x ONLY)-**
(registers 8 and 9)
<table>
<thead>
<tr>
<th>Register Value (\text{(hex)})</th>
<th>Register Value (\text{(decimal)})</th>
<th>Step Mode</th>
<th>Theoretical Resolution (\text{(^\circ)})</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>Full</td>
<td>1.8</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>Half</td>
<td>0.9</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>1/4</td>
<td>0.45</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>1/8</td>
<td>0.225</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>1/16</td>
<td>0.1125</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>1/32</td>
<td>0.05625</td>
</tr>
<tr>
<td>6</td>
<td>6</td>
<td>1/64</td>
<td>0.028125</td>
</tr>
</tbody>
</table>

Register 8 represents the positioning mode desired. That is to say, to what fractional portion of a "normal step" (full step) do you wish to resolve your motion? This register may be set directly using the \textbf{V8nn} command, or indirectly using the \textbf{Dn} or \textbf{DPn} commands (see individual command descriptions for more detailed explanation).

Register 9 (SMC50x only) represents the slewing mode desired. This represents your choice of the mode in which you wish the controller to do the majority of a commanded motion. If set the same as register 8, the motor's larger "slewing" motions will be performed entirely in the same step mode as the final positioning motions. Should register 9 be set to a lower number, any stepping other than that required to achieve the final precision position adjustment, will be done in the mode selected by the register 9 number. Register 9 may be set directly using the \textbf{V9nn} command, or indirectly using the \textbf{Dn} or \textbf{DSn} commands (see individual command descriptions for more detailed explanation).

Using the \textbf{Dn} command (example: \textbf{D6},) sets registers 8 and 9 to the same value. The \textbf{DPn} command (example: \textbf{DP4},) sets the register 8 value only . The \textbf{DSn} command (example: \textbf{DS1},) sets the register 9 value only .

-notes-
Serial Communications

SMC-40x/50x/60x controllers are easily connected to standard ASCII terminals, programmable controllers, and personal computers having RS-232-compatible ports.

Serial communications to and from the SMC units are accomplished by means of RS-232-compatible signals on connector J2. Pin 5 of J2 is the chassis/signal common (ground). Pin 2 is the data input to the SMC unit. Pin 3 is the data output from the SMC unit. Cabling to standard IBM PC™ computers and compatibles is as shown below-

Serial TTY Emulation-

If you do not have a TTY-style terminal, or should you wish to use your personal computer instead, the following BASIC-language program should allow you to communicate with SMC units successfully. A better solution is to obtain the *Motion Link™* software package available from your Kollmorgen representative, which is a full-featured communications package and program editor for the SMC-series controllers.

```
10 CLS
20 PRINT "9600 BAUD TERMINAL EMULATOR, COM1 PORT"
30 LOCATE 3,1,1,11,11
40 OPEN "COM1:9600,N,8" AS #1
50 ON COM(1) GOSUB 110
60 COM(1) ON
70 REM TRANSMIT CHARACTERS FROM KEYBOARD TO CONTROLLER
80 A$=INKEY$:IF A$="" THEN 60
90 PRINT A$;:GOTO 70
110 REM DISPLAY RECEIVED CHARACTERS FROM CONTROLLER
120 ALL=LOC(1):IF ALL<1 THEN RETURN
130 B$=INPUT$(ALL,#1):PRINT B$:;RETURN
```

If you need to use COM2 instead, simply change the references COM1 and COM(1), to COM2 and COM(2) respectively. The baud rate can also be changed to suit your SMC’s setup in line 40.
Baud Rates, Switch Settings-

The DIP switch on the front panel of the controller allows you to set up the baud rate, line termination, single or multiple "drop" mode, and the address for the controller (if multiple drop mode is selected).

⇒ For single drop (one controller per serial port) communications, it is required for the MULTI switch to be turned OFF and the TERM switch to be turned ON. The baud rate switches are set to the desired baud rate. In single drop mode communications, the state of the ADDR0-2 switches does not matter.

At the time of power up only, the switches are sampled to determine the desired configuration. Therefore, when desiring to make a change in the configuration, it is necessary to power down, change the switches, and power it back up again. The meaning of the baud rate switches are-

<table>
<thead>
<tr>
<th>SW6</th>
<th>SW7</th>
<th>SW8</th>
<th>Baud</th>
</tr>
</thead>
<tbody>
<tr>
<td>on</td>
<td>on</td>
<td>on</td>
<td>9600</td>
</tr>
<tr>
<td>off</td>
<td>on</td>
<td>on</td>
<td>4800</td>
</tr>
<tr>
<td>on</td>
<td>off</td>
<td>on</td>
<td>2400</td>
</tr>
<tr>
<td>off</td>
<td>off</td>
<td>on</td>
<td>1200</td>
</tr>
<tr>
<td>on</td>
<td>on</td>
<td>off</td>
<td>600</td>
</tr>
<tr>
<td>off</td>
<td>on</td>
<td>off</td>
<td>300</td>
</tr>
<tr>
<td>on</td>
<td>off</td>
<td>off</td>
<td>150</td>
</tr>
<tr>
<td>off</td>
<td>off</td>
<td>off</td>
<td>75</td>
</tr>
</tbody>
</table>

The switch appears as shown-

All communications to and from the SMC-series of controllers are expected to be 8 bits, no parity, 1 stop bit, with no handshaking. Configure your host device without Xon/Xoff or other handshaking for proper operations.

RS-232 standard voltage levels and data conventions are observed. Serial data input must transition greater than +/-3 volts to be recognized (+/-25 volts absolute maximum). Serial data output is driven to at least +/-5 volts.

The serial data input line, while conforming to RS-232 voltage standards, has a reduced current load. This is done so that as many as eight controllers can be connected to a single standard serial port.

As with any other RS-232 serial device, the baud rate may have to be reduced if the linking cable is very long, of high capacitance, or is located in an electrically noisy environment. There are few "hard and fast" rules, experimentation being the best guide. For example, a 150 foot run of low capacitance shielded serial cable might easily outperform 10 feet of "hookup" wire, particularly at higher baud rates.
MultiDrop Communications

The multidrop mode allows you to "daisy-chain" up to eight SMC-40x/50x/60x controllers onto a single standard RS-232C port.

The use of this mode requires that one and only one controller is configured with it's "line terminator" switch (DIP switch #5 position- "TERM") ON. All other "terminator" switches in the group must be off. In addition, all controllers in the group must have their NORM/MPX switches (DIP switch #4 position) ON, and each must be set to have a unique address within the group. All baud rate settings within the group must match one another for reliable communications.

The addresses are defined by the ADDR0 thru ADDR2 switches (DIP switch #1-#3 positions), as follows-

<table>
<thead>
<tr>
<th>SW1</th>
<th>SW2</th>
<th>SW3</th>
<th>Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>on</td>
<td>on</td>
<td>on</td>
<td>0</td>
</tr>
<tr>
<td>off</td>
<td>on</td>
<td>on</td>
<td>1</td>
</tr>
<tr>
<td>on</td>
<td>off</td>
<td>on</td>
<td>2</td>
</tr>
<tr>
<td>off</td>
<td>off</td>
<td>on</td>
<td>3</td>
</tr>
<tr>
<td>on</td>
<td>on</td>
<td>off</td>
<td>4</td>
</tr>
<tr>
<td>off</td>
<td>on</td>
<td>off</td>
<td>5</td>
</tr>
<tr>
<td>on</td>
<td>off</td>
<td>off</td>
<td>6</td>
</tr>
<tr>
<td>off</td>
<td>off</td>
<td>off</td>
<td>7</td>
</tr>
</tbody>
</table>

Hardware-wise, lines 2,3,5 and 8 should be paralleled between all J2 plugs in the group, as shown below:

Communications to a multidrop mode controller within the group must be preceded by a preamble consisting of the (STX) character (02 hex, control "B") and the address of the desired target controller (from 0-7 ASCII, that is to say from 30 hex to 37 hex). The message string can then be sent, and the controller having the matching address will listen. All others will ignore the string.

If you want ALL the controllers to listen to a message, the address sent should be "8" (38 hex), for example: (STX)8...message...

If you wish to stop in the middle of a communication to a particular controller and talk to another, transmit an (ETX) (03 hex, control "C"), and then the new preamble and other controller's address, etc.

Example of use-

ASCII-  <STX>2M+1000,<ETX>
Actual bytes sent (hex)-  02 32 4D 2B 31 30 30 30 2C 03

ASCII-  <STX>3M-2000,<ETX>
Actual bytes sent (hex)-  02 33 4D 2D 32 30 30 30 2C 03
The messages cause controller 2 to be set up to do a 1000 step clockwise move, but not to execute it yet, controller 3 is set to do a 2000 step counterclockwise move, and controller 4 is set to do a seek to absolute position 0. The last message tells all of them (address 8 is “everybody”) to execute the commands they were given, starting at the same moment.

When asking a particular controller to return a variable or counter, it will return the (STX) character, then its own address, then the data requested in the same format as in the non-multi-drop communications mode. This may be done even while other controllers in the group are busy executing commands (but not communicating over the serial link).

**Example**-

*Sent to the controller*-  
```
ASCII-                <STX>2R<LF> 
Actual bytes sent (hex)-  02 32 52 0D 
```  

*Returned from the controller*-  
```
ASCII-                <STX>2R001000<LF> 
Actual bytes received (hex)-  02 32 52 30 30 30 30 30 30 31 30 30 30 0D 
```  

**TERM Switch**-

The reason for the TERM (line terminator) switch is that in multidrop mode the transmitter portion of the controller must be shut down (so as not to load the data line) except when actually sending data. The circuitry was therefore designed to only be able to pull the data line in one polarity, and only upon specific demand. Something is needed to bias the line in the opposite direction at idle. The TERM resistor performs this task. It is not acceptable to have all the resistors on the line at once though, so the switch allows you to switch out *all but one* (preferably the farthest one from the host device in the physical cable linking the controllers together). In non-multidrop communications modes the TERM switch must be on *always* because there is only a single unit on line.
The command set is very versatile, containing many interesting and useful features. The operation of the SMC-40x/50x/60x is quite simple. Basically you send the command string you want and then a carriage return (0D hex, 13 dec) to begin executing the command sequence.

Should you wish to interrupt the command sequence in progress, send an ESC character (1B hex, 27 dec). Should you wish to re-execute the command string, just send another carriage return.

A DEL character (7F hex, 127 dec) also stops command execution, but also deletes the stored string and resets the buffer pointer.

There are 64 bytes of storage in the main command buffer, which is volatile (lost if the power is shut off). There are also seven non-volatile command buffers of 64 bytes each. These may be loaded and executed from a command string, or routines within them may be called as subroutines.

If you wish to back up the write pointer (if a mistake was made, for example), the BS (08 hex/dec) backspace character will back the pointer up one byte.

While not necessary in many instances, you should put delimiter characters between commands for clarity. They may be any non-alphanumeric characters, such as commas, periods, etcetera.

The SMC-40x/50x/60x is capable of reading numerical data inputs as hexadecimal or decimal values. It also can read variable-length numbers. For example, the following numbers within command strings are interpreted as exactly equal:

```
000008, 00008, 0008, 008, 08, 8,
```

As you can see, you can save space in the command string by truncating the numerical value entries wherever possible.

**Programming Hints-**

When you send command strings through the serial link, they are buffered in a volatile (not preserved in the event of power-down) ram buffer. These command strings can be executed immediately by sending a carriage return, or they may be stored in non-volatile (preserved) memory for later use.

Some types of command strings, notably subroutines and commands intended for execution at time of power up, must be stored into non-volatile memory using the ">>n" command where n= the buffer number (0-6). After doing a storing operation, the SMC is left in a peculiar state. That is to say, it doesn't know whether to execute the string stored, dump it, or just drop into idle (especially if the autoexecute bit is set). A lockout has been provided that prevents further operations until an ESC (to halt) or a DEL (to halt and dump) have been sent (preferably, send both).

You may examine the contents of any stored buffer by using the DEL or ESC characters to reset the buffer pointers, then enter <n and a carriage return (where n= 0-6), and finally an ENQ (05 hex/dec). The SMC will then return the newly retrieved string, suitably filtered to be printable ASCII.

**Stand-Alone Applications Programming-**
To make a stand-alone program, it is generally only necessary to store the desired program string to buffer zero (type in the string, review it with the ENQ command, and then use the \texttt{>>0} command to store it to buffer zero), set the autoexec bit with the \texttt{@1} command, and save its setting with the \texttt{>>7} command.

Upon the next power up, the SMC will dutifully load and execute whatever command string you have placed in buffer zero (including all features and options such as calls, links, and whatever else you would normally be able to do).

Hint- remember to do a \texttt{DEL} or \texttt{ESC} after each “save” command.

Power Up Defaults for Outputs-

The output lines can be caused to power up in any pattern you desire by putting them into the pattern (using \texttt{O} and/or \texttt{P} commands) and then saving the local environment ( \texttt{>>7} command).

Counted Loop Commands-

When using the "counted" loop command (\texttt{Lnnmmmm}), remember that the only times that the loop counter gets reset automatically are at the initial start of command string execution, and at the occurrence of the "K" command. An example of a pitfall to avoid would be-

\begin{center}
\begin{tikzpicture}
\draw[->] (0,0) -- (1,0);
\draw[->] (0,-1) -- (1,-1);
\node at (0.5,0) {Initial Buffer -};
\node at (0.5,-1) {Buffer 5 -};
\node at (1,0) {S100,W10,S0,L0063,<5};
\node at (1,-1) {M+200,W10,L0009,.........};
\end{tikzpicture}
\end{center}

Note that the second use of the "L" command in the second string will not loop back because the pass count has been exceeded already in the first use. There should have been a "K" command before the second "L" command as in-

\begin{center}
\begin{tikzpicture}
\draw[->] (0,0) -- (1,0);
\draw[->] (0,-1) -- (1,-1);
\node at (0.5,0) {Initial Buffer -};
\node at (0.5,-1) {Buffer 5 -};
\node at (1,0) {S100,W10,S0,L0063,<5};
\node at (1,-1) {K, M+200,W10,L0209,.........};
\end{tikzpicture}
\end{center}

Changing Step Resolutions Within Programs-

It is not usually advisable (but is possible) to change motor operating modes during a program string execution, as the step counter reflects steps in whatever mode is currently active. The step counts are not converted when the step resolution is changed. If you can live with this, change modes as often as you like. If not, you should reset the absolute position counter after making the changes. Also, after changing modes, you should plan in at least a 20 millisecond delay for the motor to settle into its new "rest" position (use a \texttt{W2} command).

\textit{There is an exception to this however.} When changing the slewing mode only on SMC-50x units, the step counts are still kept in the positioning step resolution. The accuracy of the absolute position counter is therefore unaffected.
Multidrop Polling of Controllers-

While using multidrop communications (more than one SMC unit per serial link), the asynchronous poll (SYN character) should never be used with an address of 8. This is because all the controllers on line will try to answer at once. While no damage will be done, the resulting garbled mess of data will be meaningless. Instead, poll the individual units one at a time, waiting for a response (or for a reasonable amount of time between successive attempts if none is received).

Multidrop mode polls should not occur more often than 50 times per second. This is because a multidrop poll request requires three bytes (STX, address, SYN), and the response requires four bytes (STX, address, status, carriage return), for a total of seven bytes minimum. At 9600 baud, this alone requires around 7.5 milliseconds, and the SMC requires some free processing time, especially if executing a motor motion at a high rate of speed when the poll is requested. Overloading the serial link could cause the step drive logic to “hiccup” because it has to share precious interrupt time with the serial link.

Delimiters Within Program Statements-

Please note that the use of commas or other delimiters after commands is advisable for logical reasons. Also it helps to make the program statements more understandable. If you are doing more than one command per program statement, they are required. They are also usually required to end numeric entries. Please refer to specific commands for the usage and requirements for delimiters.

-notes-
In all examples shown in this manual, "\" denotes a carriage return.
Returns the preset acknowledge byte contained in register number 5 followed by a carriage return (0D hex, 13 decimal). The acknowledge byte is generally used to signal the host computer or terminal device that a command or sequence of commands has been completed.

The power up default for this byte is the ASCII "ACK" byte (06 hex/dec), but it can easily be changed per your needs by using the V5xx command, where xxx is the value of the byte you wish returned in hex or decimal, depending upon the current controller numeric mode. The value may range from 0-FF hex (0-255 decimal). Example-

V565, !

Assuming the decimal numeric mode is active, this will cause the acknowledge byte to be set to 65 decimal, which is "A".

Should you wish to make the new value the permanent "power up" default, you should then issue the ">>7" command to store it to non-volatile memory. This will have the effect of taking a "snapshot" of the current environment settings (including this variable) and storing them to non-volatile memory.

The acknowledge byte may be changed "on the fly" at any time from within an operating command string. This will allow status or progress reporting during execution of a programmed sequence.
Form- \text{Bxx},! \text{ where } xx \text{ is a character position number}

Function- Unconditional branch to position xx within the current command string.

This command will perform an unconditional branch (it will always branch) to the specified position within the stored command string. It is the programmer’s responsibility to ensure that the position specified is correct for the purpose intended. Example-

\text{M+64,B00,}!

The example shown above will perform a 64 step move clockwise, and then loop back to position zero in the string. This repeats the move again, etcetera. Thus the motion sequence will continue until interrupted by either the reception of an ESC character (1B hex, 27 decimal) or a power-down of the board. An expanded version of this shows the character positions-

\begin{array}{cccccccc}
\text{Position in String} & 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\
\text{Stored String} & \text{M+64}, & \text{B00}, & \uparrow \\
\end{array}

Note that the position number can be in hexadecimal or decimal (as selected) and can branch to any position in the buffer (forward or backward) in the range of 00 to 3F hex (0-63 decimal). Each buffer is 64 characters long.
Cc

Format-  \texttt{Cc}[,\texttt{l}]  where c is a + or -

Function-  Sets the capture (STOP) logic to + or - edge sensitivity.

The "STOP" input serves as a hardware stop input, much the same as the serial ESC command. The Cc command sets the input to be sensitive to the positive (\texttt{C+}) or negative (\texttt{C-}) edge transition of the incoming signal. On some units this input is labeled "CAPT".

This setup condition can be changed at any time, even within an operating program string. Should you wish to make your choice the new "power up" default, issue the "\texttt{>>7}" command after using this command.

Unlike the ESC command which in addition to stopping motions or programs in progress also resets the autoexecute bit, the use of the Capture/STOP input does \textbf{not} affect the autoexecute bit in any way. The reason for this is that if an ESC is received, there must obviously be a "host" device which can decide how to proceed and then issue further commands. If the Capture input is used, there probably isn't (although there could be). Therefore it is assumed that the hardware restart inputs (see \texttt{CRn}/\texttt{CRnm} command) may be used to restart. In that instance, the autoexecute bit (if set in the first place) must be left intact so as to allow continued operations.
Format- \( C_n, \) where c is a 0 or 1

Function- To set the hexadecimal or decimal numeric modes.

Entering the \( C_0, \) command will set the hexadecimal numeric mode for data entry and readback. Entering the \( C_1, \) command sets the decimal mode. This is very convenient for program development, and allows simple interfacing to differing host controllers which may only support one mode or the other.

The use of this command affects the formats of several other commands. Among those affected are-

\[ \text{Bxx, Emmmmmm, I, Lnnmmmm, Mcmmmmmm, Pn, R, Smmmmmmm, Un, Vnmm, and Wmmmm.} \]

Please refer to the specific detailed command pages to see how each is affected. In general however, the numeric fields will be expanded by one or two digits in the decimal mode. An typical example would be the "E" command as shown:

- In Decimal mode- \( E1000, \)
- In Hexadecimal mode- \( E3E8, \)

Either of these commands presets the auxiliary position counter to 1000 decimal (3E8 hex).

For commands that return data, the length of the numeric field returned may vary between decimal and hexadecimal modes. This is because it will generally require more digits to describe a numeric value in decimal than in hexadecimal. Please refer to the specific commands for details.

Should you wish to make your choice the new "power up" default, you may enter the "\text{>>7}" command after setting the mode.
Format- \texttt{CPx,!} where \( x \) is an output line number

Function- To enable or disable the position capture output line.

This command is used to allow an output line to be used as a "position comparator" output. Allowable values range from 0-3. The upper outputs (4-7) on the -PS and -PSA units are not available to this command. This command is \textit{not available} for any of the SMC50x style units.

If active, the specified output line will be off, or "high" when the absolute position counter's value is numerically less than the "auxiliary position" register. The specified output line will be turned on, or "low" when the absolute position counter's value is greater than or equal to the "auxiliary position" register.

If the value for "\( x \)" in the command is anything other than 0-3, the feature will be turned off. The power up default is off, and it can only be enabled by specific command (cannot be saved as power up default).

This feature, when enabled is active at all times, including during motions. The counters will be compared and the specified output line will be adjusted on or off at the completion of each step.

Please note that due to the software overhead involved in the execution of this command, it is only usable up to stepping rates of about 6750 pps.
Format: \texttt{CRx,!} or \texttt{CR2n,!} where x is a restart mode number, n=input line number

Function: To select the program restart mode.

When the autoexecute bit is set (refer to the "@n" command), the stored program sequence may be stopped by either an ESC character via the serial link, or by the use of the hardware CAPTURE (STOP) input. If the SMC is to be operated in a stand-alone application (no host computer), then a suitable method of restarting the program sequence must be provided.

If the \texttt{CR0} command has been issued or the autoexecute bit has been reset, the controller will NOT restart except by explicit command via the serial link.

If the \texttt{CR1} command has been issued and the autoexecute bit has been set, the controller may be restarted by simultaneously bringing the HOME and LIMIT inputs to a logical low. Since motions would be inhibited if the two lines were left low, the controller will then wait until at least one of the two lines has been allowed to go high. At that time, normal functioning of the HOME and LIMIT inputs will resume. This mode has the advantage of being a bit safer than the CR2 mode (see next paragraph), and in addition does not occupy any of the general purpose inputs.

If the \texttt{CR2n} command is issued (where n=0-3) and the autoexecute bit is set, bringing input line "n" low will restart the stored program. This is easier to implement hardware-wise than the CR1-type function, but has the disadvantage of occupying one of the general purpose input lines. It is also not as safe "logically" since only a single input is required to restart the program sequence. Input lines 4-7 on the 401/600 -PS and -PSA units are not available for use by this command.

The ">>7" command will save your choice to non-volatile memory, thus becoming the "power up" default.
Format- \( D_c \) or \( D_{-c} \) where \( c \) is + or -

Function- Allows the motor windings to be turned off completely.

If the "D+" command is sent, one or more motor windings will always be energized, even when in an idle mode. If the "D-" command is sent, the motor windings will automatically be turned off approximately 0.2 seconds after the completion of a motion in progress (or upon the receipt of the command if already idling).

When a motion is started, the windings currents are automatically turned on for the motion, and again turned off 0.2 seconds after the completion.

If you wish to use this mode and hold an accurate position while powered down, you should use the "wave drive" mode (SMC40x/60x-series only). This is because in either the fullstep or the halfstep modes while two windings are on, the motor position will be "split" between two poles. When the windings are turned off, the motor will naturally align itself to the closest (or strongest) pole. This might cause a mis-step upon commencement of motion, and will certainly cause a "twitch" when the power is turned off. Either way, the last commanded position may not be accurately held.

It is usually not possible to accurately hold a position while powered down for the microstepping drives (SMC50x) unless you happen to stop at a pole position exactly. Worse, when the power is restored, the motor may decide to snap to the previous or next position (a full step off) from the correct microstep position. This is an unavoidable consequence of the tendency for a permanent magnet stepping motor to try to align itself to its natural pole positions at rest.

Please refer to the "Dn", "DSn" and "DPn" commands (following pages) for details of the motor stepping mode selections.
Formats- \( D_n, D_{Sn}, D_{Pn} \) where \( n \) is the motor resolution
Function- To set the motor’s drive modes.

Usage varies somewhat between the SMC-40x/60x series and the SMC-50x series of controllers, so each must be discussed separately. In all models however, the use of the "\( \gg > 7 \)" command will store your choice of mode(s) to the non-volatile memory, thus becoming the "power up" default mode. Please refer to the Drive Modes and Methods section for details about specific motor drive resolutions and methods.

SMC-40x/60x series-
The SMC-40x/60x controls use the "\( D_n \)" command only. The "\( D_{Sn} \)" and "\( D_{Pn} \)" commands will have no effect whatsoever. For either series (including -PS and -PSA versions), full, half, and wave modes are available.

The "\( D_0! \)" command selects halfstep mode, 400 ppr, 0.9° per step. Mixed mode.

The "\( D_1! \)" command selects fullstep mode, 200 ppr, 1.8° per step. Two phase on mode.

The "\( D_2! \)" command selects wavedrive mode, 200 ppr, 1.8° per step. One phase on mode.

The actual step angle resolutions will vary with the style of motor used. The data shown above are for standard 50 pole motors (200 1.8° full steps per revolution).

The motor drive mode may be changed within a command sequence if desired, but if the motor windings are energized during the change, the motor may change position slightly. This is an inherent quirk of the drive circuitry, and is unavoidable. It is therefore advised that if positioning accuracy is paramount, you should set the mode, perform a home/calibrate motion, and then proceed with sequenced motions. Once set, no uncommanded motor steps will be performed.

SMC-50x series-
The SMC-50x series (including -PS and -PSA) controllers use all three forms of the mode setting command ("\( D_n \), "\( D_{Sn} \), "\( D_{Pn} \)). This is because the -50x series controllers have the capability not only to operate in fullstep through 1/64 step resolution modes, but also to do large motions and fine positioning in different step modes automatically.

The number "\( n \)" as applied to the SMC-500/501 controllers is used as follows:

<table>
<thead>
<tr>
<th>Number</th>
<th>Mode</th>
<th>Steps/Rev</th>
<th>Angular Resolution</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>fullstep (1/1)</td>
<td>200</td>
<td>1.8°</td>
</tr>
<tr>
<td>1</td>
<td>halfstep (1/2)</td>
<td>400</td>
<td>9.9°</td>
</tr>
<tr>
<td>2</td>
<td>quarter step (1/4)</td>
<td>800</td>
<td>0.45°</td>
</tr>
<tr>
<td>3</td>
<td>eighth step (1/8)</td>
<td>1600</td>
<td>0.225°</td>
</tr>
<tr>
<td>4</td>
<td>sixteenth step (1/16)</td>
<td>3200</td>
<td>0.1125°</td>
</tr>
<tr>
<td>5</td>
<td>thirty second step (1/32)</td>
<td>6400</td>
<td>0.05625°</td>
</tr>
<tr>
<td>6</td>
<td>sixty fourth step (1/64)</td>
<td>12800</td>
<td>0.028125°</td>
</tr>
</tbody>
</table>

The actual step angle resolutions will vary with the style of motor used. The angles shown above are for standard 50 pole motors (200 1.8° full steps per revolution).

When the "\( D_{n!} \) (\( D_0, D_1 \), etcetera) command is used, both the positioning and slewing modes are set to the same mode (\( n=0-6 \)).
The "DPn!" command allows the positioning mode to be selected independently from the slewing mode. The positioning mode defines the number of microsteps the motor's "natural" or "full" steps will be split into for purposes of position resolution.

If for example, a normal "full" step for the motor used is 1.8 degrees (200 steps per motor shaft revolution), and the DP3! command is sent to the controller, positioning would then be done in 1/8 steps. These steps would be 0.225 degrees each (1600 steps per motor shaft revolution). Step counts will also be tallied at a 1600 step per revolution rate.

The "DSn" command allows the slewing mode to be selected independently from the positioning mode. The slewing mode defines the number of microsteps the motor's "natural" or "full" steps will be split into for purposes of performing large motions. This allows large motions to be performed very rapidly while maintaining final positioning accuracy and step counts in the selected positioning resolution.

The only restriction that applies (other than the 0-6 range for "n"), is that the slewing mode must be the same or coarser (lower number for "n") than the positioning mode. This is because the slewing mode of operation is by definition a larger portion of the total motion, and a less precise motion than the final adjustment motions required to achieve the desired positioning resolution.
The auxiliary position counter is useful as an alternate "seek" target or as a storage register. It is also the comparison register for the "CPx" mode of operation (position comparison).

Note that the count "mmmmmm" or "mmmmmmmm" may be of any of the following forms, all of which are equivalent, because the controller looks for up to 6 digits in the hexadecimal mode, or 8 digits in the decimal mode. It will in fact retain the last (right-most) 6 or 8 digits in the numeric field (as appropriate). Any non-alphanumeric character may be used as a delimiter for shorter numeric fields.

```
E123,
E0123,
E00123,
E000123,
E0000123,
E00000123,
```

These are ALL equivalent commands!

Obviously the fewer digits actually used, the more efficient communications will be, and the less buffer space will be used when storing a command string. This can be quite significant for complex command sequences. Note that a numeric field delimiter of some sort is required by this command.
Fnm

*Formats*- \( \text{Fnm}, \) (where \( n \) and \( m \) are input line numbers)

*Function*- Sets the SMC unit to follow pulse and direction inputs.

The Fnm command allows the general input line number "\( n \)" to be used as a CW/<CCW> control input. It also allows the general input line number "\( m \)" to be used as a *negative-edge* sensitive step trigger line. The absolute position counter will be updated while this command is performed. Also, the position comparison feature ("CPx" command) is updated after each step, if enabled.

Input line numbers "\( n \)" and "\( m \)" may range from 0-3 for non-PS/PSA units and from 0-7 for 401/600 PS/PSA units. The two numbers may not be the same (in other words, a command such as "F33" would not be accepted).

Since each number is always a single digit, a delimiter for the command is not strictly necessary, but is still advised. You must NOT use a delimiter between the "\( n \)" and "\( m \)" numbers. Valid examples of this command include-

- F01,
- F72,
- F10,
- F63,

The first one shown causes input #0 to be used as a CW/<CCW> (high = CW, low = CCW) control input and input #1 to be used as a *negative edge sensitive* step input, for instance.

This mode is canceled only by the receipt of an ESC (1B hex) character, the use of the hardware "CAPTURE" (STOP) input, or by powering down the controller.

The maximum step rate should not exceed approximately 7500 pps for maximum reliability. The direction input should be stable at least 25 microseconds prior to commanding a step, and the step input pulse's minimum width (high or low time) is 50 microseconds.

Please note that the signals should be "clean" (no bounce). Unconditioned contact closures are generally not acceptable, *especially* for the step input.
Formats- \( \text{G+! G-!} \) (where + and - indicate CW or CCW direction)

Function- Does a single step in the direction specified.

If a "G-" command is sent, the motor will go one step clockwise. A "G+" command will send the motor one step counterclockwise. The position counter is updated appropriately.

This command and repetitions of it are typically sent to do single steps for alignments or to perform slow jogging motions. It can of course be used from within a command string. An example of this would be:

\[ \text{W2,G+,B00,} \]

This command string will cause the motor to go one step clockwise every 20 milliseconds until stopped with the receipt of an ESC (1B hex, 27 decimal) character or a power-down. The HOME and LIMIT inputs are still functional during the use of this command.

The step will be performed in the currently selected positioning mode (fullstep, halfstep, quarterstep, etcetera).

The absolute position counter will be updated whenever this command is performed. Also, the position comparison feature ("CPx" command) is updated after each step, if enabled.
Formats- \( H^+ \) \( H^- \) (where + and - indicate CW or CCW direction)
Function- Commands the motor to home in a CW or CCW direction.

The receipt of an "H+" command will cause the motor to accelerate from the minimum preset speed to the maximum preset speed. This motion will continue until the "LIMIT" input line is brought low, signifying the clockwise motion limit.

The receipt of an "H-" command will commence a similar homing motion, but in the counterclockwise direction. The motion will continue until until the "HOME" input line is brought low, signifying the counterclockwise motion limit.

A typical motion velocity profile is shown below. Note that the motion stops abruptly at the time the appropriate limit is activated.

The absolute position counter is updated during any homing motions, but is not automatically zeroed at the motion limits. The position comparison feature (if available and enabled) is active during this motion.

Should you require a less abrupt ending to the homing motion, try using the Hnc command (soft home) instead.
Formats: \text{Hn+}_n \; \text{Hn-}_n \quad (\text{where } + \text{ and } - \text{ indicate CW or CCW direction, } n = \text{input line } \#) \n Function: \text{Commands the motor to soft home in a CW or CCW direction.} \n
Similar to the regular homing commands (H+, H-), these commands home CW (c=+) or CCW (c=-) until their respective limits are reached (see previous page on H+, H commands). Unlike the regular homing commands however, the soft homing commands allow the use of "soft limit" inputs to slow the motor prior to reaching the ultimate motion limits. This can be quite important if the load being moved is heavily inertial or delicate, or if high precision is required.

The "n" in the command syntax represents one of the general purpose input lines (0-3 for non-PS/PSA units, or 0-7 for 401/600 PS/PSA units). When the selected input is driven low during a homing motion, the motor speed is automatically reduced to the preset minimum speed for the remainder of the motion.

The absolute position counter is updated during a homing motion, but is not automatically zeroed at the motion limits. The position comparison mode is active (if enabled).
I

**Formats-**

\( I \) (the letter is used by itself)

**Function-** Used to return the contents of the I/O port.

The "I" command **returns** the contents of the general purpose I/O port in one of the following formats-

\[ \text{I\text{nn}} \] (hexadecimal mode)

\[ \text{I\text{nnn}} \] (decimal mode)

*The meaning of the returned data varies somewhat depending upon the unit type it is being returned from.* This is because non-PS/PSA units have only four general purpose inputs, whereas 401/600 PS/PSA units have eight. The following tables show the meaning of each data bit within the returned number field.

**For non-PS/PSA units and all 50x series-**

<table>
<thead>
<tr>
<th>Bit Number</th>
<th>Decimal</th>
<th>Hex</th>
<th>Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>input 0</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>2</td>
<td>input 1</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>4</td>
<td>input 2</td>
</tr>
<tr>
<td>3</td>
<td>8</td>
<td>8</td>
<td>input 3</td>
</tr>
<tr>
<td>4</td>
<td>16</td>
<td>10</td>
<td>output 0</td>
</tr>
<tr>
<td>5</td>
<td>32</td>
<td>20</td>
<td>output 1</td>
</tr>
<tr>
<td>6</td>
<td>64</td>
<td>40</td>
<td>output 2</td>
</tr>
<tr>
<td>7</td>
<td>128</td>
<td>80</td>
<td>output 3</td>
</tr>
</tbody>
</table>

As an example, if all the output lines were set low, inputs IN0,1, and 2 were also low, but IN3 high, the string returned by this command would be: \( \text{I}\text{F8} \). The status bits representing OUT0-3 are inverted because the output drivers themselves are inverting. In other words, when driven by a logical 1, an output will turn on, but "on" means it will clamp the output line to a low voltage. Simply think of it as a "1" in a bit position for an output as representing "on".

**For 401/600 PS/PSA units-**

<table>
<thead>
<tr>
<th>Bit Number</th>
<th>Decimal</th>
<th>Hex</th>
<th>Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>input 0</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>2</td>
<td>input 1</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>4</td>
<td>input 2</td>
</tr>
<tr>
<td>3</td>
<td>8</td>
<td>8</td>
<td>input 3</td>
</tr>
<tr>
<td>4</td>
<td>16</td>
<td>10</td>
<td>input 4</td>
</tr>
<tr>
<td>5</td>
<td>32</td>
<td>20</td>
<td>input 5</td>
</tr>
<tr>
<td>6</td>
<td>64</td>
<td>40</td>
<td>input 6</td>
</tr>
<tr>
<td>7</td>
<td>128</td>
<td>80</td>
<td>input 7</td>
</tr>
</tbody>
</table>

As an example, if all inputs were low except IN3, the string returned by this command would be: \( I08 \)
Formats- \textbf{Jnm}! (where \(n\) and \(m\) are input line numbers)

Function- Sets the controller to "jog" mode. (ESC or CAPTURE to exit)

The "\textbf{Jnm}" command puts the controller into a jogging mode which uses the input line "n" as the CW/<CCW> (hi = CW, low = CCW) direction control, and line "m" as the jog control line. As long as the "m" line is held low, the jogging motion will continue.

The controller will respond to changes in the status of the "n" direction line only when the motor motion is stopped.

The jogging mode has a rate divisor feature built in and active. This is because the motor rate starts at the minimum preset rate, and ramps up to the maximum preset rate while the "m" line is held low. If the rates were used as-is, they would be useless as a jogging mode because the normal rates (even at minimum settings) are too fast.

The solution was to provide a step rate "divisor" which divides the preset minimum and maximum rates (and everything in between) by the divisor value. That is to say that if your minimum rate is set at 120 steps per second, and you set a divisor of 40 decimal (28 hex), the minimum jogging rate will be 3 steps per second. Similarly if the maximum rate has been set at 5120 steps per second, with the same divisor, the maximum jogging rate will be only 128 steps per second. If you want very fine control, set the slope to a lesser acceleration rate value of say 15 or 20, rather than 1 or 2 (please refer to the acceleration tables).

The absolute position counter is updated during all jogging motions, and the motion limit inputs are always active. The position comparison mode is also active (if enabled). A typical motion profile is as follows-

![Typical Motion Profile](attachment:image.png)
Formats- \( \text{JPc} \) (where c is + for CW or - for CCW direction)

Function- Step (jog) at a preset exact speed until stopped.

It is sometimes useful to perform extended motions at exact stepping rates. Typical usages would include pump and feed applications. This command must be used in conjunction with the "VRnnnnnn" command, which is used to establish the target rate.

Once commanded, the motor will commence stepping clockwise (JP+) or counterclockwise (JP-) at the selected rate. There is no ramping. The selected rate is normally obtained with a high degree of precision. This is important in metering applications.

This command can only be stopped by use of an ESC (1B hex) character sent over the serial link, the triggering of the CAPTURE/STOP input, or the assertion of the HOME or LIMIT inputs (as appropriate).

This command can be used in conjunction with the MD+/ (motion rate divisor) command to achieve extremely slow or "irrational fraction" stepping rates (such as 266.333 steps per second). It can also be used in conjunction with the PV+/ (position verification) command (not available on 401/600 PS/PSA units).

Important-

When available and active, a pulse-per-revolution encoder must have signaled the completion of each full revolution, or the command stops. This pulse is expected on the CAPTURE input, and should be continuously active low (TTL/CMOS) for a period of at least two consecutive step pulses. It is preferable to have more less a 50% duty cycle as the rotation proceeds. The minimum acceptable pulse width (high or low) on this signal input is 50 microseconds.
Formats- K (the character is used by itself)
Function- Resets the loop counter.

When using the counted loop command (see the "L" command), the counter used by the command is automatically reset at the commencement of a stored program string. If the counted loop command is to be used multiple times within a command string however, it is necessary to reset the counter before each use (otherwise, the counted loop command doesn't know where to start). The examples below demonstrate the usage of this command.

G+,W4,L004,

The above string does one step clockwise, waits 40 milliseconds, and repeats the sequence four more times. Thus a total of five steps are performed. The loop counter was automatically preset at the start of the command string, so four repeats nets five total executions. Now let us look at a variation that shows the need for the "K" command-

G+,W4,L004,K,G-,W4,L124,

The above string also does five total steps clockwise, but then does five more counterclockwise. If you didn't have the "K" command in the string, it would drop through the second "L" command on the very first pass because the loop counter would already exceed the specified limit.
Formats- \textit{Lnnmmmm,,} (where \textit{nn} is the buffer position and \textit{mmmm} is a count limit)

Function- Loop to specified position "\textit{nn}" in the command buffer "\textit{mmmm}" times.

This command is used to repeat a command or group of commands a specified number of times. The "\textit{K}" command is associated with this command, and is used to zero the loop counter within a program string. The "\textit{mmmm}" count specifier is up to four or five digits in length depending upon whether the hex or decimal numeric mode is selected.

An example of use is as follows:

\begin{verbatim}
G+,W4,L004,,J
\end{verbatim}

The underlined portion is the destination address for the loop.

This command string will loop back to the start of the buffer (position zero) four times. Since the \textit{G+} and \textit{W4} commands are executed once before the \textit{L004} command, they are actually executed a total of five times (once plus four loop backs). Executing the command string automatically zeroes the loop counter prior to starting. If you wish to use more than one "\textit{L}" command in the string, you may need to re-zero the counter with the "\textit{K}" command.

Note that the "\textit{mmmm}" number is a variable-length format number, ranging from 0-FFFF or 0-65535, depending upon the numeric mode set. Thus the following commands are all equivalent:

\begin{verbatim}
G+,W4,L004, J
G+,W4,L0004, J
G+,W4,L00004, J
G+,W4,L000004, J
G+,W4,L0000004, J
\end{verbatim}

An important thing to remember is that the loop limit number must be followed by any non-alphanumeric character to act as a delimiter. Also, the "\textit{nn}" field is \textit{always} 2 digits, as its range is 0-63 (dec) or 0-3F (hex). Do not place a delimiter after the "\textit{nn}" field.

Buffer positions ("\textit{nn}" field) start from position zero, not one. All characters including delimiters count as filled positions. For example, the following command statement will loop back five times to the \textit{comma} just past the "\textit{K}" command-

\begin{verbatim}
K,G+,W4,L01005, J
\end{verbatim}

Since the comma is a non-executable character (a delimiter), this is an acceptable practice (and in fact is preferred over jumping to the exact start of a command, although you can do that if you wish).
**MC**

*Formats-*  
**MC**,

*Function*-  
Clears and initializes the indexed motion ram memory buffer area.

On -401 and -600 PS/PSA units, it is possible to use either the higher or lower group of four inputs to select from an array of up to sixteen previously stored motion commands. This command prepares the ram memory buffer area of the CPU for receiving one or more motion commands to be stored to the EEPROM (non-volatile) memory. It also has the effect of disabling all sixteen possible indexed motions. Example of usage-

```
MC, !
```

If enabled (refer to the **MM** command), this command can be performed as part of a command string at any time. This command should be performed prior to entering any indexed motion setup commands (see the **MP** command). It should also be performed at least once when setting up the controller “from the factory”. The cleared buffer is only saved to EEPROM if the indexed motion feature is enabled and the `>>?` command is issued. Use of a delimiter after the condition character is optional, but advised as being good form.

Related commands include **MI**, **MP**, and **MM**.
Mcmmmmmm

**Formats**- Mcmmmmmm,! (where c is + or - for direction, mmmmmm is a step count)

**Function**- Move mmmmmm steps CW or CCW *relative to current position*.

An M+mmmmmm command will do a clockwise move, and an M-mmmmmmm command will do a counterclockwise move. The absolute position counter is updated during the move, in the selected stepping mode.

When moving counterclockwise, if the "HOME" input is brought low, the motion will be stopped. Similarly, if in a clockwise motion, the "LIMIT" input is brought low, the motion will also be stopped. In either instance, the absolute position counter will reflect the position at the time of the stoppage.

All ramping, minimum and maximum rates, and the number of steps specified are handled automatically by the controller. There is no host intervention required past entering the command string. An example of use is:

M+4000,!

This will move the motor 4000 decimal (FA0 hex) steps clockwise from its current position. Please note that these moves are relative to the current position, not absolute. In other words, if your motor was at absolute position 5000, and you commanded it to move -3000, it would stop at position 2000 (absolute).

Similarly if your motor was at a position of 1000, and you moved -5000, the new absolute position would be FFC000 (hex mode) or 16773215 (decimal mode) (remember that the absolute position counter is 24 bits in size and would underflow at zero).

The mmmmmm data field is a variable-length field. Use any non-decimal (decimal mode) or non-hexadecimal (hex mode) character to delimit the number.

The number of steps performed will match the number commanded exactly for any stepping mode of the SMC-400x/60x series controllers including -PS/PSA units.

On any of the SMC-50x series, regardless of the positioning mode selected, if the same slewing mode has been selected, the actual number of steps performed will match the number commanded. If however, the slewing mode has been specified to be different (always must be the same or coarser), the number of "steps" actually performed will not necessarily be the same physical number of steps as was commanded. In this instance, the number of logical steps performed will match the number commanded, and the position counters will reflect the actual position in terms of the *resolution mode* selected. This is, in essence, the automatic mode indexing feature of the SMC-500 at work.

The MD+/- (motion rate divisor) and CPx (position comparison) commands will function with this command if so desired (SMC-400,401 only).
**Formats:**  \( \text{Mcc},! \) (where cc is ++ or -- for direction)

**Function:** To initiate a continuous, unlimited motion.

The SMC controllers normally perform motions of specific step count. There may be situations where you wish to instead perform continuous motions for specific periods of time, or just continuous slewing motions without stops.

The \( \text{M++} \) and \( \text{M--} \) commands work exactly as the \( \text{M+/-nnnnnn} \) command previously described, except that there is no stopping count limit. The motion continues until an ESC (1B hex) character is sent, the appropriate limit input is brought low (LIMIT for CW motions or HOME for CCW motions), or the STOP/CAPTURE input is brought low.

Regardless of motion duration, the absolute position counters are kept current throughout the motion (see \( \text{M+/-nnnnnn} \) command for notes concerning SMC-50x auto mode indexing).

On the SMC-400 and 401 **only**, the \( \text{MD+/-} \) (motion rate divisor) and \( \text{CP+/-} \) (position comparison) commands will function in conjunction with this command.
**MDc**

*Formats-* \( \text{MDc,} \, c \) (where \( c \) is + for “on” and - for “off”)

*Function-* Divides *all motions* by the jog rate divisor.

It is sometimes necessary to use extremely slow or irrational (such as 33.3333... per second) stepping rates. Normally the jog rate divisor register (see "U" and "V" commands) becomes active only during the jogging mode. The SMC-series controllers have the ability to cause the divisor to be active for all motions. These include the “jog at preset rate” (JP), seek and move commands. Example of usage-

\[
\text{MD+} \, \uparrow
\]

(turn the motion divisor on)

This command must be performed from a command line (it cannot be stored as a power up default). It can be performed as part of a stored command string and automatically executed, however.
**MI**

**Formats** - \(\text{MI}_n\) (where \(n\) is 0 or 1)

**Function** - Causes an indexed motion to be performed.

On -401 and -600 PS/PSA units, it is possible to use either the higher or lower group of four inputs to select from an array of up to sixteen previously stored motion commands. For this command to be used, several preconditions must be met. First, the operating mode must have been previously enabled (refer to the \text{MM} command). Second, the motion parameters must have been previously loaded into the CPU's ram memory either by means of a \(<7\) command (fetching them from EEPROM), or by specifically setting up the motions with the \text{MP} command. Third, the motion parameter selected by the group of inputs (see below) must be a valid entry. Examples of usage-

\[
\text{MI0}, .\]
\[
\text{MI1}, .\]

If enabled, the number \(n\) is used to select the lower inputs 0-3 (\(n=0\)) or upper inputs 4-7 (\(n=1\)). The selected input group is sampled, and their numeric representation is used to select from one of 16 possible motions (including direction) stored in the ram memory. The selections must have individually been set up prior to executing this command (refer to the \text{MP} command). After determining that the motion selected by the input line combination is valid, it will be performed.

Let us assume that the input pattern looks like-

<table>
<thead>
<tr>
<th>Input0-</th>
<th>Input1-</th>
<th>Input2-</th>
<th>Input3-</th>
<th>Input4-</th>
<th>Input5-</th>
<th>Input6-</th>
<th>Input7-</th>
</tr>
</thead>
<tbody>
<tr>
<td>low</td>
<td>low</td>
<td>high</td>
<td>low</td>
<td>low</td>
<td>low</td>
<td>high</td>
<td>high</td>
</tr>
</tbody>
</table>

Using the \text{MI0} command will cause the 5\(^{th}\) stored motion to be done, whereas the \text{MI1} command will cause the 12\(^{th}\) motion to be performed (assuming that they have been preset).

Because these motions are performed from local CPU memory and not from EEPROM, they are executed very quickly and because there are no laborious input test/branch statements to be parsed, their start timing is extremely uniform. The usual execution time for this command is around 75-80 microseconds exclusive of the motion time. The range on any given motion is +/-16,777,215 steps. The motion performed is an otherwise normal motion with all the features of any other commanded motion.

Related commands include \text{MC}, \text{MP}, and \text{MM}.
Formats: \texttt{MMc,} \texttt{c} \texttt{(where c is + for "on" and - for "off")}

Function: Enables or disables the indexed motion feature.
(SMC-401PS/PSA and SMC-600PS/PSA only).

On -401 and -600 PS/PSA units, it is possible to use either the higher or lower group of four inputs to select from an array of up to sixteen previously stored motion commands. This command enables (+) or disables (-) the entire feature, including the saving and retrieval of the motion commands to/from the onboard EEPROM memory. Example of usage -

\begin{itemize}
  \item \texttt{MM+,} \texttt{c} \texttt{(turn the indexed motion feature on)}
  \item \texttt{MM-,} \texttt{c} \texttt{(turn the indexed motion feature off)}
\end{itemize}

The result of this command must be saved using the \texttt{>>7} command to be saved as a power up default. It can be performed as part of a command string at any time, however. Use of a delimiter after the condition character is optional, but advised as being good form.

Related commands include \texttt{MI}, \texttt{MP}, and \texttt{MC}. 
Formats: \( \text{MPnnccmmmm} \) (where \( nn \) is a motion number, \( c \) is +/- direction, and \( mmmmm \) is the number of steps)

Function: Causes an indexed motion to be set up in the CPU's ram memory.

On -401 and -600 PS/PSA units, it is possible to use either the higher or lower group of four inputs to select from an array of up to sixteen previously stored motion commands. To use this feature, the mode must be enabled (\text{MM+} command) and each desired motion must have been previously defined using this command. Examples of usage:

\[
\begin{align*}
\text{MP5+1234, J} \\
\text{MP11-20446, J}
\end{align*}
\]

(decimal mode assumed for this example)

The first example defines motion number 5 as +1234 steps. The second defines motion number 11 as being -20446 steps.

The numeric range of \( nn \) is 0-15 decimal (0-F hex). The range of \( mmmmm \) is 0-16,777,215 decimal (0-FFFFFFF hex). The \( c \) entry must be either + (CW) or - (CCW).

The normal sequence of operation is to use the \text{MC} command to clear the ram buffer and all the enables first, then use this command to define any or all of the motions. After defining one or more motions, the \text{MI} command may be used to perform them.

Only those motions specifically defined will be enabled. In other words, if you only define motions 0, 2 and 6 and then try to do a motion number 9 using the \text{MI} command, nothing will happen. The CPU knows which motions have been defined (valid).

Once defined, the motions may be executed at will, or they may be saved to the EEPROM non-volatile memory using the \text{$\gg$7} command. If saved, they will be reloaded to the CPU's ram memory for use at power up or upon execution of the \text{$<$7} command.

Even if automatically loaded at power up or from the \text{$<$7} command, they may be overwritten at any time using this command, including from within program strings. This allows for great flexibility in using the indexed motion feature.

Important: Since these parameters occupy the same memory space as the "call" buffer, it is extremely important not to use this command after having called a subroutine, or from a subroutine. You must first either re-enter the desired motion definitions using the \text{MC} command and this command, or perform a \text{$<$7} to retrieve previously stored definitions.

Related commands include \text{MC}, \text{MI}, and \text{MM}.
Formats- \textit{N} (the character is used by itself) 
Function- Returns the auxiliary position counter. 

This command is used to interrogate the auxiliary position counter that was preset by the "E" command. The format of the returned data is as follows- 

\textsc{Nmnmnmnm} 

The auxiliary position counter/register is useful to store a commonly used "target" position for logical seeks. When you wish to seek the auxiliary position, you merely need to issue the "T" command instead of seeks or moves. 

Alternatively, it contains the number to which the absolute position counter is compared if the \textit{CPx} command is used (SMC-40x only, no -PS/PSA units). 

Finally, it may also be used as a 24-bit storage register for whatever purpose you wish.
Onc

Formats- Onc_c (where n is an output line number and c is + or -)
Function- Sets output line "n" high (c = +) or low (c = -).

This command sets output line "n" high (c = +) or low (c = -). The outputs are open collector-driver types. Thus the command to turn a particular output low actually turns the output driver on.

The line number can range from 0-3 for non-PS/PSA units and all 50x units, and 0-7 for 401/600 PS/PSA units.

Note that the ">>7" command will save the current output pattern to non-volatile memory, thus becoming the "power up" default condition. This is extremely useful to preset outputs driving valves, lamps, etc. upon power up.
Formats- \textit{Pnn!} (where \( nn \) is a pattern in numeric format)

Function- Sets all the outputs to pattern \( nn \) simultaneously.

This command allows all four general purpose outputs to be simultaneously set to the pattern described by the number. A "0" in a particular bit of \( nn \) turns the corresponding output driver "on". A "1" in a particular bit of \( nn \) turns the corresponding output driver "off". The output drivers are open-collector types. The applicable format of number \( nn \) varies somewhat by unit type.

\textbf{For non-PS/PSA units-}

<table>
<thead>
<tr>
<th>Bit Number</th>
<th>Decimal</th>
<th>Hex</th>
<th>Destination</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>output 0</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>2</td>
<td>output 1</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>4</td>
<td>output 2</td>
</tr>
<tr>
<td>3</td>
<td>8</td>
<td>8</td>
<td>output 3</td>
</tr>
<tr>
<td>4</td>
<td>16</td>
<td>10</td>
<td>-none-</td>
</tr>
<tr>
<td>5</td>
<td>32</td>
<td>20</td>
<td>-none-</td>
</tr>
<tr>
<td>6</td>
<td>64</td>
<td>40</td>
<td>-none-</td>
</tr>
<tr>
<td>7</td>
<td>128</td>
<td>80</td>
<td>-none-</td>
</tr>
</tbody>
</table>

As an example, performing a \textit{P12!} command in decimal mode would set outputs 0 and 1 on (low) and outputs 2 and 3 off (high).

\textbf{For PS/PSA units-}

<table>
<thead>
<tr>
<th>Bit Number</th>
<th>Decimal</th>
<th>Hex</th>
<th>Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>output 0</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>2</td>
<td>output 1</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>4</td>
<td>output 2</td>
</tr>
<tr>
<td>3</td>
<td>8</td>
<td>8</td>
<td>output 3</td>
</tr>
<tr>
<td>4</td>
<td>16</td>
<td>10</td>
<td>output 4</td>
</tr>
<tr>
<td>5</td>
<td>32</td>
<td>20</td>
<td>output 5</td>
</tr>
<tr>
<td>6</td>
<td>64</td>
<td>40</td>
<td>output 6</td>
</tr>
<tr>
<td>7</td>
<td>128</td>
<td>80</td>
<td>output 7</td>
</tr>
</tbody>
</table>

As an example, performing a \textit{P64!} command in decimal mode would set all outputs on (low) except for output 6, which would be off (high).

Note that the "\textit{>>7}" command will save the current output pattern to non-volatile memory, thus becoming the "power up" default condition. This is extremely useful to preset outputs driving valves, lamps, etc. upon power up.
Formats- \texttt{PVc,c} (where c is + or - for on or off)

Function- To control the position verification mode (SMC-40x ONLY).

The SMC-40x (other than -PS/PSA) controllers have the ability to interface with a pulse-per-revolution encoder or other switch. This encoder or input should be connected to the CAPTURE input, and should be reasonably well debounced (optical sensors are preferred). PV+ will enable the feature, PV- will turn it off.

This command is usable only with the "JP" (jog at preset rate) command, and is intended for metering or feeding applications.

When enabled, the motor will be allowed to move up to move up to 225 steps (if in fullstep mode) or 450 steps (if in halfstep mode) before seeing a high-to-low transition of the CAPTURE input. If a transition is seen within this count, the motion will continue. If it isn't seen, the "JP" command will cease operation. Refer to the JPC command for more details.

This command is not available on SMC-50x units or 401PS/401PSA/600PS/600PSA units.
Formats- Qc,c (where c is + or - for on or off)
Function- To enable or disable the auto power reduction mode.

The "Q+" command will enable the auto power reduction mode, and the "Q-" command will disable it. The auto power reduction mode, if enabled, will automatically reduce the winding currents to approximately 50% of the running current at idle. This reduction takes place about 0.2 seconds after the end of a motion or after receipt of the command if already stopped at idle. The current level is automatically returned to 100% of the front panel setting at the start of a motion.

The reason for the delay is to allow the motor and the attached mechanism to decelerate from the last minimum-velocity step, to a stable rest state. The power reduction mode, as its name implies, reduces the power dissipation of both the motor and controller when maximum torque is not necessarily needed.

On SMC-500/501 models only, if the slewing and positioning modes are set differently, the auto power-down mode is automatically turned off. This is due to an inherent tendency for the motor to seek the nearest even, aligned pole position when the power is reduced. In the auto mode, indexing modes, and especially when the selected slew mode is full or half step, the position the motor relaxes toward, may in fact overshoot the next slew mode step position! When this occurs, the motor can experience severe resonance problems, and may run backwards, at multiples of the driven step rate, or may stall immediately. Shutting it off prevents these problems. This is an inherent trait of permanent magnet steppers and cannot be avoided except by maintaining full winding power.

The chosen condition may be made the "power up" default by issuing the ">>7" command after making your choice.
**Formats**- \( R! \) (the letter is used by itself)

**Function**- Returns the absolute position counter.

This command is used to interrogate the absolute position counter. The absolute position counter is updated during any and all motions, no matter what the motion may be. Its contents represent the current position of the motor in terms of the number of steps from an arbitrary "zero" point. This "zero" point is established by use of the "Z" command, usually after a homing motion. The format of the returned data is as follows-

\[ Rnnnnnnn! \]

**Other Examples**-

A returned value of 000100 (hex) or 00000256 (decimal) represents an absolute position of 256 steps clockwise from the zero point.

A returned value of FFFF00 (hex) or 16777115 (decimal) represents an absolute position of 100 steps counterclockwise from the "zero" point.

The absolute position counter is always kept current in the selected positioning mode. For all SMC-400/401/600 units, this is the selected motor mode. For SMC-500/501 units, it is the selected positioning mode.
**Formats:** \texttt{Smmmmmm,...} (where mmmmmm is an absolute position step count)

**Function:** Performs a logical seek to the desired \textit{absolute} position.

This command is extremely useful for positioning command sequences. The user, rather than having to figure out the number of steps and direction to go to get to a desired position, needs only to use the \texttt{"S"} command with the target position specified. The controller then automatically determines the direction and number of steps necessary to get there. Note that the controller will do the shortest possible move to the target position \textit{within the 24-bit counter range}.

This means that if the current absolute position is for example: 000300, and an \texttt{"S600,..."} command is sent, the motor will be moved 300 steps clockwise. If the command \texttt{"S100,..."} is sent instead, it would move 200 steps counterclockwise.

If the absolute position is currently FFFF00 (100 hex steps CCW from "home"), and the \texttt{"S00600"} command is sent, the motor will go counterclockwise FFF900 (hex) steps because this is the shortest possible move within the counter's range.

\textit{The absolute position step counter is always assumed to be positive with respect to "zero".} While this may seem a bit odd to users more familiar with servo motions (typically having a +/- range from "zero"), it is in fact quite straightforward and logical.
Formats- \text{ T!} \quad \text{(the character is used by itself)}

Function- Causes the motor to logically seek the auxiliary step position count.

This command will cause the motor to logically seek the position count stored in the auxiliary register. This register should have been preset using the "E" command prior to using this command. Note that the controller will do the shortest possible move to the target position \textit{within the 24-bit counter range}.

This means that if the current absolute position is for example: 000300, and the auxiliary position is 000600, when the "T" command is sent, the motor will be moved 300 steps clockwise. If the auxiliary position was 000100 instead, it would move 200 steps counterclockwise.

If (in the hex mode, for example) the absolute position is currently FFFF00 (100 hex steps CCW from "home"), and the auxiliary position is 000600, when the command is sent, the motor will go counterclockwise FFF900 (hex) steps because this is the shortest possible move within the counter's range. The same principles apply to the decimal numeric mode.

\textit{The absolute position step counter is always assumed to be positive with respect to "zero"} (please refer to the "S" command).
**Un** or **Vnmm**

*Formats-*  
**Un,**! or **Vnmm,**!  
(see text)

*Function-*  
These commands return or set the value of stored variables.

These commands are presented together since they are closely related. Only the data (mm or mmm) field changes in length between the hex and decimal modes.

The "**Un,**" command is used to *return* the value of variable number "n". The "**Vnmm,**" or "**Vnmmm,**" command is used to *set* the value of variable number "n" (except for number 6, which is the GSTAT register, and is read-only).

The registers and their usages are as follows-

<table>
<thead>
<tr>
<th><strong>N</strong></th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Minimum rate pointer</td>
</tr>
<tr>
<td>1</td>
<td>Maximum rate pointer</td>
</tr>
<tr>
<td>2</td>
<td>Acceleration slope</td>
</tr>
<tr>
<td>3</td>
<td>Deceleration slope</td>
</tr>
<tr>
<td>4</td>
<td>Jog rate divisor</td>
</tr>
<tr>
<td>5</td>
<td>Acknowledgement byte</td>
</tr>
</tbody>
</table>
| 6     | Contents of the GSTAT register  
D0- Motor running  
D1- CW (high) or <CCW> (low)  
D2- Clock running  
D3- Jog mode active  
D4- General timeout  
D5- Idle power mode active  
D6- Stored command readback request  
D7- Message received |
| 7     | Contents of the input port register |
| 8     | Current positioning mode (50x-series only) |
| 9     | Current slewing mode  
(50x-series only) |
The jog preset (JP) command is used when you need an exact stepping rate. This can range from 10 to 10,000 steps per second. Since the MD+/ command is also active during the JP command, extremely slow or irrational fraction step rates can be achieved. The VRnnnnn command is used to specify the desired stepping rate for the "JP" motion.

This command is simple to use, the nnnnn being the desired rate (in hex or decimal, as applicable) of steps per second you wish to "jog" continuously at. An example of usage is-

VR1234,JP+,

The motor will run at 1,234 steps per second until specifically stopped.

If the jog rate divisor register (see "V" and "U" commands) contains 121 and the MD+ command is issued, the actual stepping rate will be 10.198 steps per second. If you obtain a submultiple of the step rate you specified, check the status of the MD+/- command and turn it off or modify the jog rate divisor.
Wmmmm

*Formats-* \( \text{Wmmmm, \text{mm}} \) (where \( \text{mm} \) is the desired delay in 0.01 second increments)

*Function-* Delay of "mmmm" multiples of 10 milliseconds.

The wait command is used to space other commands or motions apart in time. The "W" command operates in 0.010 second increments, from a minimum of 10 milliseconds (mmmm or mmmmm = 1), to a maximum of 655.35 seconds (mmmm = FFFF or mmmmm = 65535).

This command is interruptable only by an ESC (1B hex, 27 decimal) character, the CAPTURE (STOP) input, or a power-down.
**Formats-** X\texttt{xxnc},\texttt{xx} (xx = buffer position, n is an input number, c is + or -)

**Function-** Branches to buffer position "xx" if input "n" is high (+) or low (-)

This command allows conditional branching within a program string, based upon the condition (high or low) of a specified input line.

The "xx" portion of the command refers to the character position within the stored program string. The "n" represents the input line number (0-3 for non-PS/PSA units, 0-7 for 401/600 PS/PSA units only). The "c" must be either a "+" of the branch true condition is to be upon line "n" high, or a "-" if the branch true condition is to be line "n" low.

\texttt{G+,W2,X002-,n,}

The underlined text is the target buffer position for the branch.

The above example program string will do a step every 20 milliseconds in the clockwise direction for as long as the input line number 2 is held low.

Note that whether the hex or decimal numeric modes have been selected, the "xx" field remains two characters, as the ranges are 0-3F and 0-63 respectively. The command buffer is 64 characters in length.
**Yn**

*Formats:* \( \text{Yn},! \) \((n = \text{a group of preset data values, 0-7 or "Y"})\)

*Function:* Resets selected variables to a group of preset values.

If you should desire to reset the stored parameters listed below to the defaults shown below, use the "Y" command. The "YY" command resets the controller to the factory defaults.

They are reset as follows:

**SMC40x-**

<table>
<thead>
<tr>
<th>n=</th>
<th>(0)</th>
<th>(1)</th>
<th>(2)</th>
<th>(3)</th>
<th>(4)</th>
<th>(5)</th>
<th>(6)</th>
<th>(7)</th>
<th>(Y)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Register number-</td>
<td>(0) minimum rate</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>4</td>
<td>8</td>
<td>12</td>
</tr>
<tr>
<td></td>
<td>(1) maximum rate</td>
<td>15</td>
<td>31</td>
<td>47</td>
<td>63</td>
<td>79</td>
<td>95</td>
<td>111</td>
<td>127</td>
</tr>
<tr>
<td></td>
<td>(2) acceleration slope</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>(3) deceleration slope</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>(4) jog rate divisor</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>(5) acknowledgement byte</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
</tr>
</tbody>
</table>

**SMC50x-**

<table>
<thead>
<tr>
<th>n=</th>
<th>(0)</th>
<th>(1)</th>
<th>(2)</th>
<th>(3)</th>
<th>(4)</th>
<th>(5)</th>
<th>(6)</th>
<th>(7)</th>
<th>(Y)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Register number-</td>
<td>(0) minimum rate</td>
<td>2</td>
<td>2</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>8</td>
<td>10</td>
<td>12</td>
</tr>
<tr>
<td></td>
<td>(1) maximum rate</td>
<td>15</td>
<td>31</td>
<td>47</td>
<td>63</td>
<td>79</td>
<td>95</td>
<td>111</td>
<td>127</td>
</tr>
<tr>
<td></td>
<td>(2) acceleration slope</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>(3) deceleration slope</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>(4) jog rate divisor</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>(5) acknowledgement byte</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
</tr>
</tbody>
</table>

All values shown are decimal values.

The results can then be saved to non-volatile memory (becoming the new power up defaults) by the use of the ">>7" command.
**Formats**  
Z...Z
ts (the letter is used by itself)

**Function**  
Resets the absolute position counter to zero.

This command may be used at any time to reset the absolute position counter. It is typically used after homing motions. It should be used prior to any usage of the logical seek-style motion commands ("T" and "S").

Please note that while the absolute position counter is automatically set to 000000 upon power up, subsequent "homing" motions **do not** automatically reset the counter upon reaching their respective "home" limits.
<STX> (control B)

**Formats** - Single character of 02 hex/decimal value, no carriage return follows it

**Function** - Start flag for a multidrop message

Used to signal all SMC controllers on a multidrop line to listen for the selector address which must follow this character in a multidrop message sequence.

Typical example of use:

ASCII message- `<STX>4M+1000,`!

Actual bytes sent (hex)- 02 34 4D 2B 31 30 30 2C 0D

This message instructs the controller at address 4 to move 1000 steps clockwise, and all others to ignore the message entirely. Please refer to the section on multidrop communications for further details of the multidrop communications mode.
<ETX> (control C)

Formats- Single character of 03 hex/decimal value, no carriage return follows it

Function- NON-carriage return termination of a multidrop message communication string.

Should you wish to terminate a multidrop command or communication message string without causing the listener to execute what has already been sent, you should terminate the string with the ETX character. This is highly useful to synchronize motor controllers within a multidrop environment.

Example of use-

<table>
<thead>
<tr>
<th>ASCII-</th>
<th>&lt;STX&gt;2M+1000,&lt;ETX&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>Actual bytes sent (hex)-</td>
<td>02 32 4D 2B 31 30 30 30 2C 03</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ASCII-</th>
<th>&lt;STX&gt;3M-2000,&lt;ETX&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>Actual bytes sent (hex)-</td>
<td>02 33 4D 2D 32 30 30 30 2C 03</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ASCII-</th>
<th>&lt;STX&gt;4S0000,&lt;ETX&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>Actual bytes sent (hex)-</td>
<td>02 34 53 30 30 30 30 2C 03</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ASCII-</th>
<th>&lt;STX&gt;8,</th>
</tr>
</thead>
<tbody>
<tr>
<td>Actual bytes sent (hex)-</td>
<td>02 38 0D</td>
</tr>
</tbody>
</table>

The messages cause controller 2 to be set up to do a 1000 step clockwise move, but not to execute it yet, controller 3 is set to do a 2000 step counterclockwise move, and controller 4 is set to do a seek to absolute position 0. The last message tells all of them (address 8 is “everybody”) to execute the commands they were given starting at the same moment.
Formats- Single character of 05 hex/decimal value, no carriage return follows it
Function- Returns the current active command string.

The ENQ character may be sent at any time the controller is idling (that is to say that the stored program is not executing). It is useful either to review the currently buffered string, or to examine what has been sent do far while entering a program string.

The ENQ command causes the command string in the “active buffer” to be sent back preceded by an ENQ character, and followed by a carriage return. A typical returned string is as follows:

Contents of ACTIVE buffer- M+1000,W20,L003-,

Returned string (ASCII)- <ENQ>M+1000,W20,L003-,

Returned bytes (hex)- 05 4D 2B 31 30 30 30 2C 57 32 30 2C 4C 30 30 33 2C 0D

Please note that this is an external-only command, and may not be stored or executed as part of a program string.
**<SYN> (control V)**

**Formats**- Single character of 16 hex (22 decimal) value, no carriage return follows it.

**Function**- To perform an asynchronous poll of a controller.

It is often useful to be able to interrogate the controller at random to determine run/stop, direction, and program execution status.

The SYN command is an external command (cannot be part of a command string) which causes the SMC unit to return its current status in the form of a single byte followed with a carriage return (non-multidrop mode). In the multidrop mode, the standard multidrop mode return protocol is followed (please refer to the multidrop communications section for details). Examples of usage is as follows-

**Normal Mode**-

Command sent- `<SYN>` (this is actually just the SYN character, which is 16 hex, 22 decimal)

Returned reply- `N!` (see below for the format of `N`)

**Multidrop Mode**-

Command sent- `<STX>n<SYN>` (where `n` is the address of the controller to be polled)

Returned reply- `<STX>nN!` (see below for the format of `N`)

**Format of `N`**-

D0- Motion in progress
D1- CW (high) or <CCW> (low) motion currently
D2- Command string being executed
D3- always zero
D4- always one
D5- always one
D6- always zero
D7- always zero

`N` therefore ranges in value from 30 hex through 37 hex ("0" to "7" ASCII).

This command may be issued at any time, up to 50 times per second if desired, however please note that due to the software overhead involved in the execution of this command, it is only usable up to stepping rates of about 8750 pps (40x/60x-series) or 5500 pps (50x series).
Formats- Single character of 1B hex (27 decimal) value, no carriage return follows it
Function- Interrupts any motion or command in progress.

Can be used by the host device or operator to stop or interrupt any motion or command in progress. This constitutes a software stop interrupt.

The escape character is an external command, and may not be stored or executed as part of a command string.

The escape character resets the autoexecute enable bit. This is because it is assumed that if the escape command is received, there is a host device capable of deciding what to do to recover from the stop condition and issue new commands. In the mean time, the SMC will not take it upon itself to restart.
If the SMC controller is to be used as a standalone control device, or if it is simply desired not to have to reload the command string(s) at each power up, the autoexecute enable bit may be set. The use of the ">>7" command can then be used to make it the power up default.

If the bit is reset, the SMC will power up normally and wait for instruction strings to be sent from the host device, even if command strings are stored in its non-volatile buffers. If an ESC character (software stop interrupt) has been issued, the autoexecute bit will be reset and the controller will not proceed until told specifically what to do. The ESC command will not however reset the autoexecute enable bit stored in the non-volatile memory. It is therefore still available at the time of the next power up.

If the autoexecute bit is set at power up, the contents of non-volatile buffer zero will be loaded to the active command buffer and execution of its contents will begin immediately. Naturally this mandates buffer zero having been preloaded with the desired command string (see >>n command).

If it is set after power up, the user need then only specify which buffer to execute (see <n command), and the controller will begin automatic operations. Thus, the controller can operate as a stand-alone entirely, or can execute stored strings upon command. See the appendices for further details and examples.

As implied above, this command may be issued even from within a command string if so desired, but on the other hand it will not become the power up default unless the ">>7" command is performed.
Format- \( =\text{nmm} \) (where \( n=\)buffer number, \( mm=\)position with the buffer)

Function- To call subroutines from within buffer \( n \).

This is a very useful command, as it allows many different routines to access a particular "common" command string segment. For example, if a routine is used several different places in a command string or from within different buffers, rather than have to keep repeating the commands, a common one can be "re-used".

The "calling" command itself (\( =\text{nmm} \)) can be located at any position in any command buffer, volatile or not. However, the subroutine must itself be stored in a non-volatile buffer. There may be multiple subroutines per stored buffer, since the routine's address within the buffer is specified in this command. ALL subroutines must end with the \(^\wedge\) character (which is the equivalent of "return").

A typical usage is shown below, and in greater detail in the appendices. Each "call" requires approximately 35 milliseconds before the "called" string can be executed. Subroutines cannot be nested (in other words, subroutines cannot call other subroutines) because there is only a single storage buffer for the command string that called the subroutine.

\[
\begin{align*}
\text{Buffer 2 contents:} & \quad \text{M+1000,}=300,\text{B00}, \ldots \ldots . \\
\text{"call"} & \quad \text{"return"} \\
\text{Buffer 3 contents:} & \quad \text{W40,}^\wedge \ldots \ldots . 
\end{align*}
\]

Assuming that the buffer 2 contents are executed, the program will branch from the "=300" statement to the subroutine at the 00th position of buffer 3, which will execute and return back to the next statement in buffer 2 and resume execution from that point.

Important- On 401PS/PSA and 600PS/PSA units, when the motion index mode is enabled, the <7 command also retrieves the default preset indexed motion parameters from non-volatile memory (see \( \text{MI}, \text{MP}, \text{MM} \) and \( \text{MC} \) commands for details). Since these parameters occupy the same space as the "call" buffer, it is extremely important not to use <7 from within a called subroutine. If done, the original command string (the one the subroutine was called from) will be lost from the command buffer (it can be re-loaded from non-volatile memory). Conversely, if you wish to use the \( \text{MI}, \text{MP}, \) and \( \text{MC} \) commands, you must be aware that calling a subroutine overwrites the indexed motion parameters that are currently in the ram buffer. They can of course be retrieved with the <7 command.
^ (carat mark)

**Format-** ^ (used by itself from within a program statement)
**Function-** To "return" from a subroutine to the routine that called it.

This is the **required** delimiter for all subroutines. The function of this command is to restore the command string (and the character position within that string) from which the subroutine was **called**. The "calling" command string need not necessarily have been in non-volatile memory, although all subroutines must be stored in non-volatile memory. This is due to memory size limitations.

**DO NOT** put this command character at the end of non-subroutine command sequences. If a subroutine is executed directly rather than being properly called, the result upon encountering the "^" character is unpredictable since there is no valid return address. At power up, this address is cleared to a default "dummy" value for safety.

The return command requires only a few microseconds to execute, so there is no discernable delay between its issue and the **next** command in the "calling" string.

For examples of the use of this command, see the "=nmm" command, or the appendices concerning subroutines and programming techniques.

**Important-** On 401PS/PSA and 600PS/PSA units, when the motion index mode is enabled, the <7 command also retrieves the default preset indexed motion parameters from non-volatile memory (see **MI, MP, MM** and **MC** commands for details). Since these parameters occupy the same space as the "call" buffer, it is extremely important not to use <7 from within a called subroutine. If done, the original command string (the one the subroutine was called from) will be lost from the command buffer (it can be re-loaded from non-volatile memory). Conversely, if you wish to use the **MI, MP, and MC** commands, you must be aware that calling a subroutine overwrites the indexed motion parameters that are currently in the ram buffer. They can of course be retrieved with the <7 command.
Format-  \(<n \rangle \)  (where \( n \) is a buffer number 0 through 7)
Function- Loads command buffer \( n \) and points to its start.

Since the SMC-series controllers have only 64 bytes of volatile (immediate, active) command buffer memory, it is impossible for a lengthy command sequence to be loaded and executed without the continual intervention of a "host" computer to keep reloading command segments. In addition, it would be impossible to make the SMC units a "stand-alone", pre-programmed device without some source for the required command segments.

The solution to this was to provide non-volatile (EEPROM) buffers for command string storage. If at power up the autoexecute bit is not set, no actions are taken by the controller (other than to automatically load the power up default registers and motor setups). If the autoexecute bit is set, the previously stored buffer number \( \text{zero} \) is automatically retrieved and executed just as if you had loaded it in at the moment of power up. This makes possible complete stand-alone applications using the controller as a controlling device, or to at least have it know automatically at power up how to interact with the other parts of the control system.

If the command string you desire is too big or cumbersome to fit into a single buffer, it can be split up into smaller segments, those segments stored away into the various buffers, and then the entire command string can be executed.

When executing this command and \( n=0 \) through 6, command buffers are retrieved, when \( n=7 \), the power up defaults are retrieved. It is therefore possible to retrieve the power up defaults for the motor mode, speeds, accelerations and so forth even from within a stored command string. This can be useful.

Examples of the use of this command (and its associated "\( >>n \)" storage command) can be found in the appendices.

This command can be used at any time, directly or from within an executing program string.

Important- On 401PS/PSA and 600PS/PSA units, when the motion index mode is enabled, the \(<7 \) command also retrieves the default preset indexed motion parameters from non-volatile memory (see \( \text{MI, MP, MM} \) and \( \text{MC} \) commands for details). Since these parameters occupy the same space as the "call" buffer, it is extremely important not to use \(<7 \) from within a called subroutine. If done, the original command string (the one the subroutine was called from) will be lost from the command buffer (it can be re-loaded from non-volatile memory). Conversely, if you wish to use the \( \text{MI, MP, and MC} \) commands, you must be aware that calling a subroutine overwrites the indexed motion parameters that are currently in the ram buffer. They can of course be retrieved with the \(<7 \) command.
Format- 

>>n  (where n is a buffer number 0 through 7, NO CARRIAGE RETURN)

Function- 

To store a buffer or setup to non-volatile memory.

This command is an external only command, and cannot be executed or stored within a command string. Its purpose is to store the current contents of the command buffer (if n=0-6) or the current controller's setup information (if n=7).

If the current command buffer contents are to be stored (n=0-6), the issuance of this command copies exactly the current command buffer contents into the specified non-volatile register. The buffer should be reviewed by the use of the ENQ (05 hex) command before storing, just to be certain that it is correct. Upon issuance of this command, the following will be returned:

```
<br> <Storing> ***************************************************************<Done><br>  <------------------------ 64 total ------------------------><br>
```

Each asterisk represents a single command byte being stored. This process requires about 1.5 seconds. After doing this, you should send a DEL character, a CR, and then an ESC character. This is to completely reset the buffer pointers and prepare for another save (if any).

If n=7, then the current setup registers (including the output bit pattern) are stored to act as the new power up defaults. This includes all the programmable registers (rates, slopes, ack byte, etc., the decimal/hex mode, the capture/stop slope, the auto power-down mode, autoexecute bit, and restart options bits). When n=7, only a <STORING><DONE> message is returned, as no command bytes are being stored.

**Important-** On 401PS/PSA and 600PS/PSA units, when the motion index mode is enabled, the >>7 command also saves the preset indexed motion parameters from ram memory (see MI, MP, MM and MC commands for details). These become power-up defaults and are loaded into the ram memory upon startup and whenever the <7 command is used (even from within a stored command string).
Format- \( c \) (where c is + to enable or - to disable)

Function- To turn on or off the "Snoopy" mode.

The "Snoopy" mode allows you to echo the SMC controller's internal stepping pulses to the output line OUT 0. The direction is not output. Use the "P" (pattern) or "O" (output) command prior to making a move to signal the direction of motion on a different output line.

These pulses are active high, with the "on time" being about 35 microseconds. Each pulse represents one internal step being performed by the controller.

On the SMC-40x/60x series, each output pulse is coincident with each step performed by the controller, regardless of the step mode selected.

On the SMC-50x, each pulse is coincident with each physical step performed. When the slewing and positioning modes are selected the same, each pulse represents one step in the selected positioning mode. If the slewing mode has been selected to be coarser than the positioning mode, each pulse represents a physical step performed. These physical steps will usually represent a multiple of the selected "positioning" mode steps. See the "M+/nnnnnn" command for further details of the SMC-50x auto mode index feature, and how stepping is performed under these conditions.

The main purpose for the "Snoopy" mode is to monitor actual stepping frequency or count when desired. It may also find used to slave "pulse and direction"-type controllers to a "master" intelligent controller. It is typically necessary to stretch the pulse width for this purpose. Use of a monostable multivibrator is a common solution.

Please note that due to the software overhead involved in the execution of this command, it is only usable up to stepping rates of about 8750 pps (40x/60x-series) or 5500 pps (50x series).
Format- Single character of 7F hex (127 decimal) value.
Function- To interrupt ANY motion or command being executed.

This character can be sent at any time by the host device, in order to interrupt ANY and all motions, commands, or command sequences being executed by the SMC controller. This constitutes a software "emergency stop" function, similar to the STOP/CAPTURE input line or ESC character which are capable of stopping any motion in progress.

*This command character also has the function of resetting the command string read pointer to the start of any command string in the command buffer.* Thus, with the issue of a single carriage return, the programmed sequence that was stopped by the DEL character, may be resumed from its start.

The use of this control character will reset the autoexecute bit. If you wish to resume automatic program execution, you will need to set the bit again (please refer to the "@" command).
Format- A single question mark and a carriage return.

Function- To interrogate the current firmware revision.

This command is seldom used, but nonetheless handy to determine the controller's firmware (ROM-stored operating software) revision level and last upgrade date. It also returns the controller's model number. Simply send a question mark and a carriage return to use this command.

The format of the returned string may vary somewhat with the particular revision level and features, but is similar to:

SMC-401/SMC-600PSA Revision 1.02, 12/10/96
The stepping rates vary somewhat from SMC-40x/60x units to SMC-50x units. This is because of the larger software overhead incurred in microstepping operations. The following tables show the step rates for a register 0 (minimum) and 1 (maximum) settings. The rates shown are for a jog/motion rate divisor of 1, or jog/motion rate divisor disabled.

### Stepping Rates

The stepping rates vary somewhat from SMC-40x/60x units to SMC-50x units. This is because of the larger software overhead incurred in microstepping operations. The following tables show the step rates for a register 0 (minimum) and 1 (maximum) settings. The rates shown are for a jog/motion rate divisor of 1, or jog/motion rate divisor disabled.

#### Step rates for SMC-40x/60x only-

<table>
<thead>
<tr>
<th>Register</th>
<th>Minimum Rate</th>
<th>Maximum Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>16</td>
<td>86</td>
</tr>
<tr>
<td>1</td>
<td>94</td>
<td>87</td>
</tr>
<tr>
<td>2</td>
<td>173</td>
<td>88</td>
</tr>
<tr>
<td>3</td>
<td>251</td>
<td>89</td>
</tr>
<tr>
<td>4</td>
<td>330</td>
<td>90</td>
</tr>
<tr>
<td>5</td>
<td>409</td>
<td>91</td>
</tr>
<tr>
<td>6</td>
<td>487</td>
<td>92</td>
</tr>
<tr>
<td>7</td>
<td>566</td>
<td>93</td>
</tr>
<tr>
<td>8</td>
<td>644</td>
<td>94</td>
</tr>
<tr>
<td>9</td>
<td>723</td>
<td>95</td>
</tr>
<tr>
<td>10</td>
<td>802</td>
<td>96</td>
</tr>
<tr>
<td>11</td>
<td>880</td>
<td>97</td>
</tr>
<tr>
<td>12</td>
<td>959</td>
<td>98</td>
</tr>
<tr>
<td>13</td>
<td>1037</td>
<td>99</td>
</tr>
<tr>
<td>14</td>
<td>1116</td>
<td>100</td>
</tr>
<tr>
<td>15</td>
<td>1195</td>
<td>101</td>
</tr>
<tr>
<td>16</td>
<td>1273</td>
<td>102</td>
</tr>
<tr>
<td>17</td>
<td>1352</td>
<td>103</td>
</tr>
<tr>
<td>18</td>
<td>1431</td>
<td>104</td>
</tr>
<tr>
<td>19</td>
<td>1509</td>
<td>105</td>
</tr>
<tr>
<td>20</td>
<td>1588</td>
<td>106</td>
</tr>
<tr>
<td>21</td>
<td>1666</td>
<td>107</td>
</tr>
<tr>
<td>22</td>
<td>1745</td>
<td>108</td>
</tr>
<tr>
<td>23</td>
<td>1824</td>
<td>109</td>
</tr>
<tr>
<td>24</td>
<td>1902</td>
<td>110</td>
</tr>
<tr>
<td>25</td>
<td>1981</td>
<td>111</td>
</tr>
<tr>
<td>26</td>
<td>2059</td>
<td>112</td>
</tr>
<tr>
<td>27</td>
<td>2138</td>
<td>113</td>
</tr>
<tr>
<td>28</td>
<td>2217</td>
<td>114</td>
</tr>
<tr>
<td>29</td>
<td>2295</td>
<td>115</td>
</tr>
<tr>
<td>30</td>
<td>2374</td>
<td>116</td>
</tr>
<tr>
<td>31</td>
<td>2453</td>
<td>117</td>
</tr>
<tr>
<td>32</td>
<td>2531</td>
<td>118</td>
</tr>
<tr>
<td>33</td>
<td>2610</td>
<td>119</td>
</tr>
<tr>
<td>34</td>
<td>2688</td>
<td>120</td>
</tr>
<tr>
<td>35</td>
<td>2767</td>
<td>121</td>
</tr>
<tr>
<td>36</td>
<td>2846</td>
<td>122</td>
</tr>
<tr>
<td>37</td>
<td>2924</td>
<td>123</td>
</tr>
<tr>
<td>38</td>
<td>3003</td>
<td>124</td>
</tr>
<tr>
<td>39</td>
<td>3081</td>
<td>125</td>
</tr>
<tr>
<td>40</td>
<td>3160</td>
<td>126</td>
</tr>
<tr>
<td>41</td>
<td>3239</td>
<td>127</td>
</tr>
<tr>
<td>42</td>
<td>3317</td>
<td>128</td>
</tr>
</tbody>
</table>

84
Step rates for SMC-50x only:

<table>
<thead>
<tr>
<th>Step</th>
<th>Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>50</td>
</tr>
<tr>
<td>1</td>
<td>100</td>
</tr>
<tr>
<td>2</td>
<td>150</td>
</tr>
<tr>
<td>3</td>
<td>200</td>
</tr>
<tr>
<td>4</td>
<td>250</td>
</tr>
<tr>
<td>5</td>
<td>300</td>
</tr>
<tr>
<td>6</td>
<td>350</td>
</tr>
<tr>
<td>7</td>
<td>400</td>
</tr>
<tr>
<td>8</td>
<td>450</td>
</tr>
<tr>
<td>9</td>
<td>500</td>
</tr>
<tr>
<td>10</td>
<td>550</td>
</tr>
<tr>
<td>11</td>
<td>600</td>
</tr>
<tr>
<td>12</td>
<td>650</td>
</tr>
<tr>
<td>13</td>
<td>700</td>
</tr>
<tr>
<td>14</td>
<td>750</td>
</tr>
<tr>
<td>15</td>
<td>800</td>
</tr>
<tr>
<td>16</td>
<td>850</td>
</tr>
<tr>
<td>17</td>
<td>900</td>
</tr>
<tr>
<td>18</td>
<td>950</td>
</tr>
<tr>
<td>19</td>
<td>1000</td>
</tr>
<tr>
<td>20</td>
<td>1050</td>
</tr>
<tr>
<td>21</td>
<td>1100</td>
</tr>
<tr>
<td>22</td>
<td>1150</td>
</tr>
<tr>
<td>23</td>
<td>1200</td>
</tr>
<tr>
<td>24</td>
<td>1250</td>
</tr>
<tr>
<td>25</td>
<td>1300</td>
</tr>
<tr>
<td>26</td>
<td>1350</td>
</tr>
<tr>
<td>27</td>
<td>1400</td>
</tr>
<tr>
<td>28</td>
<td>1450</td>
</tr>
<tr>
<td>29</td>
<td>1500</td>
</tr>
<tr>
<td>30</td>
<td>1550</td>
</tr>
<tr>
<td>31</td>
<td>1600</td>
</tr>
<tr>
<td>32</td>
<td>1650</td>
</tr>
<tr>
<td>33</td>
<td>1700</td>
</tr>
<tr>
<td>34</td>
<td>1750</td>
</tr>
<tr>
<td>35</td>
<td>1800</td>
</tr>
<tr>
<td>36</td>
<td>1850</td>
</tr>
<tr>
<td>37</td>
<td>1900</td>
</tr>
<tr>
<td>38</td>
<td>1950</td>
</tr>
<tr>
<td>39</td>
<td>2000</td>
</tr>
<tr>
<td>40</td>
<td>2050</td>
</tr>
<tr>
<td>41</td>
<td>2100</td>
</tr>
<tr>
<td>42</td>
<td>2150</td>
</tr>
</tbody>
</table>
Subroutines

The SMC-series of controllers have the ability to "call" subroutines that have been previously saved to the non-volatile memory buffers. The basic method is that the contents of the current command buffer are saved temporarily to a "shadow" buffer, along with the address of the next command in the string. The specified buffered string is then retrieved from the non-volatile memory, and the command read pointer is set to the value specified in the "call" instruction. Execution of the stored string begins from that point. When the "^" character is encountered, the command string stored in the "shadow" buffer is reloaded to the command buffer, and the read pointer is set to the address of the next instruction. Program execution continues onward from that point.

Since the "subroutine" string may be retrieved essentially an infinite number of times, it can be called as often as needed, and by different portions of the command string(s). In addition, there is no distinction made as to the origin of the calling string (volatile or non-volatile memory), so subroutines can be called from any program string.

There is only one "shadow" buffer and only one pointer storage register, so subroutines cannot be nested since the ultimate return address and original calling string would be lost.

An example of typical usage is shown below. Note that the same subroutine could just as easily been called both times, and that more than one subroutine can reside in a stored buffer.

```
"calling" command string
Z,H0,-=400,W50,R,L0009,=410,B00, 
M+10,W1,M-8,^,-----,S1000,W10,T,^
```

(position in buffer 4 (hexadecimal numbering shown here))

**Important-** On 401PS/PSA and 600PS/PSA units, when the motion index mode is enabled, the <7 command also retrieves the default preset indexed motion parameters from non-volatile memory (see MI, MP, MM and MC commands for details). Since these parameters occupy the same space as the "call" buffer, it is extremely important not to use <7 from within a called subroutine. If done, the original command string (the one the subroutine was called from) will be lost from the command buffer (it can be re-loaded from non-volatile memory). Conversely, if you wish to use the MI, MP, and MC commands, you must be aware that calling a subroutine overwrites the indexed motion parameters that are currently in the ram buffer. They can of course be retrieved with the <7 command.
Command Buffers and Memory

Your SMC controller is equipped with one transient (volatile) and seven “permanent” (non-volatile) command buffers. The transient buffer is used to store incoming characters from the serial link, and as a place to load selected non-volatile buffers when it is desired to operate from their contents. All command strings must be loaded into the transient (program) buffer to be executed.

Since the transient buffer (RAM) is only 64 bytes long, it sometimes presents limitations as to the complexity of the command string that can be executed. Worse, since its contents are random at power up (indeed it must be specifically blanked out at initialization), it is useless for standalone applications where there may not be a host device to give it commands.

The solution to this problem was to provide the user with the ability to save program strings to “more or less” permanent (EEPROM) memory. When the autoexecute bit has been set (and itself stored into the non-volatile memory), the controller is capable of executing extremely complex command sequences automatically upon power up, and without any host being present to command it. A facility has been provided also to link buffers together in any order desired.

Storing Data to Non-Volatile Buffers-

First you must enter the command string you wish to save into the normal buffer (just send it over the serial link, but do NOT end it with a carriage return). It is a good idea to send a DEL or ESC first though, to reset the pointers and delete any program string previously in the buffer. Once loaded into the buffer, you enter the command >>n (no carriage return required), where n is the number 0-6, corresponding to the buffer number you want to save it to.

The controller will respond with a “start” prompt, a string of asterisks (corresponding to the actual bytes being saved), and a “done” at the end. Once saved, you can retrieve the string as often as you with the <n command (where n is the buffer number).

If, at the time a retrieve command is issued, the autoexecute enable bit has been set, the newly retrieved command string will be immediately executed. The autoexecute bit must also have been saved to non-volatile memory (>>7 command) for the power up start to work. The power up autoexecute only works from buffer zero.

The save commands (>>n) are themselves transient and cannot be executed as part of a command string, nor stored. The reason for this is that the non-volatile memories are write-limited, and as such cannot be repeatedly written to (even by accident, such as from an errant program). Any given location in the EEPROM memory may be written to about 10,000 times before it will no longer “remember”. This is of course far beyond most applications’ requirements.

Reading Data from Non-Volatile Buffers-

The retrieve commands, on the other hand, are unlimited. In this manner, any program string, transient or not, can link to any other of the non-volatile buffered strings, in any order. The main considerations to bear in mind are these:

a) The newly loaded command strings will only continue executing (like the strings that linked to them), if the autoexecute bit has been set. This can of course be set within the original program string, if so desired. In other words, if you program a statement like: S100,PA,....,<2, and the autoexecute bit has not been set, the controller will load the contents of buffer two, but will not continue execution. This can be useful if you want the program to stop and wait for the receipt of a
carriage return to continue. If however, the autoexecute bit is set, the newly loaded string (from buffer two) will be immediately executed, as if you had just typed it in and sent a carriage return.

b) If you are trying to set up a self-sufficient, stand-alone control system, you must have both stored the autoexecute bit as set, and you must have stored at least the initial command string in buffer zero. This is because the controller only knows to load buffer zero at power up if the autoexecute bit is set). Naturally the contents of buffer zero may simply be an immediate link to another buffer (if so desired), but that is your choice.

c) While it is possible to link to any of the stored buffers from any command string in the transient buffer, it is not possible to link back to a non-stored string. That is to say that if you store several command strings to various buffers, and then type in a command to link to one of them, it is not possible for any of them to link back to the string you typed in. This is of course, because it was not stored permanently.

**Auto Execute Upon Power Up**

The autoexecute-upon-power up feature makes "turnkey" controller applications possible with the SMC controllers. As mentioned on the previous page, you must load at least the initial program string into buffer zero (it can link to others as necessary), and the autoexecute bit must have been saved as "set".

In a stand-alone mode of operation, a method of starting and stopping the stored program must be provided. Since there is assumed to be either no host capable of communications, or at least no links to one, the preferred method is to have hardware inputs and outputs to interface with the rest of the system, or to operator pushbuttons, or ???. The controller has a busy output which indicates that a motion is in progress. It also has a hardware "Capture" (STOP) input, but no specifically labeled "Start" input. The choice of the lines to be used is up to you.

Should you not want restart capability at all (or do not want standalone capability), use the CR0 command to disable the mode entirely. Your choice can then be saved for power up default by use of the >>7 command. You can also selectively use this command at any time from within a program string if there are critical or safety-related portions of the running cycle that should not be restarted at all.

Should you want the capability of restarting the stored programs, you have basically three choices. The first is to send another <0 command and a carriage return (assuming that the autoexecute bit is still set). The second is to use the CR1 command, and the third is to use the CR2n command.

The CR1 option (which can also be executed at any time, and/or stored for use as a power up default), requires that to restart the programs, BOTH the <LIMIT> and <HOME> lines be brought simultaneously to a logical zero, and then one or both released. Since it is an unnatural condition for both of these inputs to be low at the same time, their simultaneous activation is a reasonable signal. Diode or switch logic can accomplish this, while maintaining full functionality of the limit inputs during motions. The fact that motions in either direction cannot be done while both are held low, mandates that at least one of them be brought high again before program execution can be started.

The other option, CR2n, basically designates one of the lower four general-purpose input lines (0-3) as the "restart" input. This may be simpler to implement in some applications, but does tie up an input that may be needed for something else. Once again, this option may be done "on the fly" at any time (even from within a program string), or may be save as the power up default.

In the stand-alone, autoexecute mode, the controller has full capabilities, just as with a downloaded host-resident program mode. The only significant difference is that you have one less command buffer available for program storage. In the non-autoexecute "host" mode, you have the full 64 bytes of transient buffer, plus the 448 bytes of non-volatile buffer can have been previously saved with callable subroutines and linkable strings. In the stand-alone mode, obviously you can't power up with anything usable in the transient buffer.
One manufacturer of the non-volatile memory device we use specifies a minimum of 10 years data retention, a second source specifies (believe it or not) 100 years. How long it will really last, we don't know, but it's probably safe to assume something between the two.

Memory Organization-

As can be seen above, the SMC's working memory is completely separate from the non-volatile memory. Command sequences that you wish to save to one of the seven non-volatile buffers (0-6) must be first input to the regular working (command ) buffer. The >>n command is then used to save the working register's contents to buffer number "n".

Since the non-volatile memory is EEPROM (electrically eraseable, programmable read-only memory), it can only be "saved" a few thousand times before it begins to "forget". For this reason, command buffer "saves" cannot be a regular part of a running program.

On the other hand, you can retrieve (read) the memory a more or less infinite number of times. Use the <<n command (even from within a stored command string) to copy a non-volatile buffer's contents to the working register. If the autoexecute bit has been previously set (by using the @1 command), the program string will execute just as if you had typed the string contents and sent a carriage return. You may also call subroutines from the NVRAM buffers at any time.
SMC-40x/50x/60x Motion Calculations

The SMC400/500/600 controllers accelerate in terms of steps per second per step. While not time-linear at higher stepping rates, close timing approximations can be made using the following formulas. The finish units are as noted.

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Source/Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>TACC</td>
<td>Time required for the acceleration portion of the motion.</td>
</tr>
<tr>
<td>TDEC</td>
<td>Time required for the deceleration portion of the motion.</td>
</tr>
<tr>
<td>NACC</td>
<td>Number of steps in the acceleration ramp.</td>
</tr>
<tr>
<td>NDEC</td>
<td>Number of steps in the deceleration ramp.</td>
</tr>
<tr>
<td>TSLEW</td>
<td>Time required for the high speed portion of the motion.</td>
</tr>
<tr>
<td>TTOTAL</td>
<td>Time required for the total motion.</td>
</tr>
<tr>
<td>VMAX</td>
<td>Maximum programmed stepping rate.</td>
</tr>
<tr>
<td>VMIN</td>
<td>Minimum programmed stepping rate.</td>
</tr>
<tr>
<td>RACC</td>
<td>Acceleration rate in steps/sec/step.</td>
</tr>
<tr>
<td>RDEC</td>
<td>Deceleration rate in steps/sec/step.</td>
</tr>
<tr>
<td>NTOTAL</td>
<td>Total number of steps for the motion.</td>
</tr>
</tbody>
</table>

For the number of steps in an acceleration or deceleration ramp (step count):

\[
N_{ACC} = \frac{V_{MAX}^2 - V_{MIN}^2}{2 (R_{ACC})}
\]

\[
N_{DEC} = \frac{V_{MAX}^2 - V_{MIN}^2}{2 (R_{DEC})}
\]

For acceleration and deceleration times (in seconds):

\[
T_{ACC} = \frac{V_{MAX} - V_{MIN}}{R_{ACC}}
\]

\[
T_{DEC} = \frac{V_{MAX} - V_{MIN}}{R_{DEC}}
\]

For the slew (high speed) motion time (in seconds):

\[
T_{SLEW} = \frac{N_{TOTAL} - (N_{ACC} + N_{DEC})}{V_{MAX}}
\]

For the total motion time (in seconds):

\[
T_{TOTAL} = T_{SLEW} + T_{ACC} + T_{DEC}
\]

The minimum and maximum rates and acceleration slopes are obtained from the tables previously listed in this manual.
The following chart shows the usable ASCII character set and their associated hexadecimal and decimal values. The format is: Character | Hex Value | Decimal Value

<table>
<thead>
<tr>
<th>Character</th>
<th>Hex Value</th>
<th>Decimal Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>NUL</td>
<td>00 00</td>
<td>+ 2B 43</td>
</tr>
<tr>
<td>SOH</td>
<td>01 01</td>
<td>, 2C 44</td>
</tr>
<tr>
<td>STX</td>
<td>02 02</td>
<td>- 2D 45</td>
</tr>
<tr>
<td>ETX</td>
<td>03 03</td>
<td>. 2E 46</td>
</tr>
<tr>
<td>EOT</td>
<td>04 04</td>
<td>/ 2F 47</td>
</tr>
<tr>
<td>ENQ</td>
<td>05 05</td>
<td>0 30 48</td>
</tr>
<tr>
<td>ACK</td>
<td>06 06</td>
<td>1 31 49</td>
</tr>
<tr>
<td>BEL</td>
<td>07 07</td>
<td>2 32 50</td>
</tr>
<tr>
<td>BS</td>
<td>08 08</td>
<td>3 33 51</td>
</tr>
<tr>
<td>HT</td>
<td>09 09</td>
<td>4 34 52</td>
</tr>
<tr>
<td>LF</td>
<td>0A 10</td>
<td>5 35 53</td>
</tr>
<tr>
<td>VT</td>
<td>0B 11</td>
<td>6 36 54</td>
</tr>
<tr>
<td>FF</td>
<td>0C 12</td>
<td>7 37 55</td>
</tr>
<tr>
<td>CR</td>
<td>0D 13</td>
<td>8 38 56</td>
</tr>
<tr>
<td>SO</td>
<td>0E 14</td>
<td>9 39 57</td>
</tr>
<tr>
<td>SI</td>
<td>0F 15</td>
<td>: 3A 58</td>
</tr>
<tr>
<td>DLE</td>
<td>10 16</td>
<td>; 3B 59</td>
</tr>
<tr>
<td>DC1</td>
<td>11 17</td>
<td>&lt; 3C 60</td>
</tr>
<tr>
<td>DC2</td>
<td>12 18</td>
<td>= 3D 61</td>
</tr>
<tr>
<td>DC3</td>
<td>13 19</td>
<td>&gt; 3E 62</td>
</tr>
<tr>
<td>DC4</td>
<td>14 20</td>
<td>? 3F 63</td>
</tr>
<tr>
<td>NAK</td>
<td>15 21</td>
<td>@ 40 64</td>
</tr>
<tr>
<td>SYN</td>
<td>16 22</td>
<td>A 41 65</td>
</tr>
<tr>
<td>ETB</td>
<td>17 23</td>
<td>B 42 66</td>
</tr>
<tr>
<td>CAN</td>
<td>18 24</td>
<td>C 43 67</td>
</tr>
<tr>
<td>EM</td>
<td>19 25</td>
<td>D 44 68</td>
</tr>
<tr>
<td>SUB</td>
<td>1A 26</td>
<td>E 45 69</td>
</tr>
<tr>
<td>ESC</td>
<td>1B 27</td>
<td>F 46 70</td>
</tr>
<tr>
<td>FS</td>
<td>1C 28</td>
<td>G 47 71</td>
</tr>
<tr>
<td>GS</td>
<td>1D 29</td>
<td>H 48 72</td>
</tr>
<tr>
<td>RS</td>
<td>1E 30</td>
<td>I 49 73</td>
</tr>
<tr>
<td>US</td>
<td>1F 31</td>
<td>J 4A 74</td>
</tr>
<tr>
<td>SPACE</td>
<td>20 32</td>
<td>K 4B 75</td>
</tr>
<tr>
<td>!</td>
<td>21 33</td>
<td>L 4C 76</td>
</tr>
<tr>
<td>&quot;</td>
<td>22 34</td>
<td>M 4D 77</td>
</tr>
<tr>
<td>#</td>
<td>23 35</td>
<td>N 4E 78</td>
</tr>
<tr>
<td>$</td>
<td>24 36</td>
<td>O 4F 79</td>
</tr>
<tr>
<td>%</td>
<td>25 37</td>
<td>P 50 80</td>
</tr>
<tr>
<td>&amp;</td>
<td>26 38</td>
<td>Q 51 81</td>
</tr>
<tr>
<td>,</td>
<td>27 39</td>
<td>R 52 82</td>
</tr>
<tr>
<td>(</td>
<td>28 40</td>
<td>S 53 83</td>
</tr>
<tr>
<td>)</td>
<td>29 41</td>
<td>T 54 84</td>
</tr>
<tr>
<td>*</td>
<td>2A 42</td>
<td>U 55 85</td>
</tr>
</tbody>
</table>

-Notes-
### General information
- Overall view 2
- Motor drive 2
- Power source 2
- Software 3

### Mounting and Installation
- Non-PS/PSA units 4
- PS/PSA units 4

### Electrical considerations
- Power Sources 6
- Wiring and ground loops 7
- Conclusion 8

### Connections and Pinouts
- Items common to all units 9
- Non-PS/PSA units only 9
- PS/PSA units only 9
- Locations of connectors 10

### Inputs and Outputs in General
- Inputs 11
- Outputs 11
- Serial I/O 11
- I/O Schematics 12

### Drive Modes and Methods
- Differences between units 13
- SMC-40x unit modes 13
- SMC-50x unit modes 14
- SMC-60x unit modes 16
- Drive methods, notes 16
- Motor connections 18
- Drive stability 19

### Registers
- Allowable values 20
- Minimum/Maximum rates 20
- Accelerations/Decelerations 20
- Jog Rate divisor 21
- Programmable ACK byte 21
- Motor resolutions (-50x only) 21

### Serial Communications
- Cabling, connections 23
- Serial TTY emulation 23
- Baud rates, switch settings 24

### Multidrop Communications
- Address settings 25
- Switch settings, cabling 25
- Example of usage 25
- TERM switch usage 26

### Command Set
- Notes 27
- Programming Hints 27
- Stand-Alone applications 27
- Power up default outputs 28
- Counted loop commands 28
- Changing step resolutions 28
- Multidrop polling 29
- Delimiters within statements 29

### Command Listings
- A (return ACK) 31
- B (branch) 32
- Cc (STOP edge selection) 33
- Cn (numeric mode select) 34
- CPx (position comparator) 35
- CRx (restart modes) 36
- Dc (motor winding power) 37
- DN,DSn,DPn (resolutions) 38
- E (set auxiliary count) 40
- F (follower mode) 41
- G (single step) 42
- Hc (home motion) 43
- Hnc (soft home motion) 44
- I (return inputs) 45
- J (jog mode) 46
- JPc (precision rate motion) 47
- K (reset loop counter) 48
- L (counted looping) 49
- M (move, relative position) 51
- MC (motion index clear) 50
- Mcc (move continuously) 52
- MDC (motion divisor) 53
- MI (perform indexed motion) 54
- MM (en/dis motion index) 55
- MP (define indexed motion) 56
- N (return auxiliary count) 57
- O (set/reset an output) 58
- P (set an output pattern) 59
- PVC (position verification) 60
- Q (auto power reduction) 61
R (return position counter)  62
S (seek, absolute position)  63
T (seek, auxiliary count)  64
U (return a register value)  65
V (set a register value)  65
VR (set precision step rate)  66
W (delay)  67
X (conditional branch)  68
Y (preset variables)  69
Z (zero position counter)  70
<STX> (multidrop preamble)  71
<ETX> (multidrop escape)  72
<ENQ> (inquiry)  73
<SYN> (asynchronous poll)  74
<ESC> (command interrupt)  75
@ (autoexecute enable)  76
= (call a subroutine)  77
^ (return from a subroutine)  78
<n (retrieve buffer contents)  79
>>n (save buffer contents)  80
\c (snoopy mode control)  81
<DEL> (command clear)  82
? (return version)  83

Stepping Rates
SMC-40x/60x rate table  84
SMC-50x rate table  85

Subroutines
General usage  86

Command Buffers and Memory
General  87
Storing data to buffers  87
Reading data from buffers  87
Auto execute at power up  88
Memory organization  89

Motion Calculations
General formulae  90

ASCII Conversion Chart
Listing  91