Price: $25.00

SLO-SYN® WARPDRIVE™
SS2000PCi
PROGRAMMABLE
STEP MOTOR
CONTROLLER
INSTALLATION
AND
OPERATION
INSTRUCTIONS

ENGINEERING CHANGES

We reserve the right to make engineering refinements on all its products. Such refinements may affect information given in instructions, Therefore, **USE ONLY THE INSTRUCTIONS THAT ARE PACKED WITH THE PRODUCT.**

RECORD OF REVISION		
Revision	Date	Description
A	7/19/99	Initial Release
B	2/3/00	Revise corporarate ID and logos.

Table of Contents

SECTIONS & TITLE **PAGE**

SECTIONS & TITLE **PAGE**

SECTIONS & TITLE **PAGE**

(This Page left intentionally Blank)

Section 1

Introduction

1.1 - HOW TO USE THIS MANUAL

Congratulations on the purchase of your new Superior Electric SLO-SYN® WARPDRIVE™ motion control product! The SS2000PCi programmable motion controller is a full-featured and flexible product, yet it is fairly simple to apply it to your machine control application. This manual is designed to guide and assist you through the installation, programming, and operation of the controller/drive. If you're reading this, you understand the importance of familiarizing yourself with how this product should be installed and operated. We strongly recommend that you read through this manual until you are comfortable with electrical connections and operating concepts of the unit. Also, for your safety, we strongly recommend that you read "Section 2 - Important Safety Information" first, then read the "Quick Start Installation Guide" section. This will provide you with the basics on how to properly wire and connect the unit into your system. From there you can move on to the "Motion Controller Programming Interface" and "Software Reference Guide" sections to learn how to program your controller/drive to suit your application. "The "Glossary" section describes the terms most commonly used in this manual. Detailed technical information is provided in the "Hardware Specifications" section.

1.2 – FEATURES AND FUNCTIONS

The SS2000PCi single axis step motor controller is a fully programmable digital indexer (controller). The indexer is a powerful controller which allows motion programming using the Motion Controller Programming Interface (MCPI). The MCPI is a Windows® based Graphical User Interface (GUI) which runs on a PC and facilitates system programming in an easy to use BASIC like language.

1) Microcontroller Based Digital Controller Card.

The controller/indexer circuit card is based on a sophisticated digital microcontroller chip. The microcontroller performs all necessary tasks for executing complex user programs including control of digital inputs and outputs (I/O), stepper motor current profiles, two serial communications ports, drive section interface, digital encoder inputs for closed loop stepper, etc.

Some of the key features of the controller are:

- High-performance motion controller uses 16-bit, 16-MHz microprocessor
- Surface-mount construction utilizing custom integrated circuits
- Easy setup and programming with Windows interface and BASIC-like language
- Simpler program construction – user specifies own motion units, e.g. inches, the programming environment automatically converts to motor "steps"
- Feature-rich command set, with over 85 functions in the following groups: Motion, Trajectory Parameters, Drive Parameters, I/O Control, Timer, Program Flow Control, Interrupts, Boolean/Relational Operators, String Handling, Variables, and Arithmetic
- Two independent serial ports: Host port RS232 or RS485; Auxiliary port is RS485; selectable communication rates up to 38.4 kbaud
- Programmable Inputs: 8 optically isolated 5-24 VDC; 8 more logic inputs (or used for BCD switches)
- Programmable Outputs:4 optically isolated 5-24 VDC, 250 mA; 4 more "sinking" open collector (or used for BCD switches)
- Built-in 12 VDC power supply for opto-coupled I/O
- Analog input: 0-10 Volts, 10-bit resolution with multiple programmable functions
- I/O on shielded 25-pin "D" connector; optional terminal-strip adapter available
- Encoder input of up to 2 million counts per second on 9-pin "D" connector
- Closed-loop modes for stall detection, position verification, and correction
- 3 LED indicators for Power, Fault, and Motion Busy

2) SS2000PCi Power Supply

- Wide range AC input 95 – 265 VAC, 50/60 Hz
- Fused AC Input
- Built-in Line Filter
- Current Foldback on DC Power Ouputs

1.3 - WHAT YOU NEED TO KNOW FIRST

This instruction manual is written in a simple and easy-to-follow format that should be suitable for both new and experienced motion control users. In order to get the most out of your WARPDRIVE Programmable Motion Controller, we assume the user will be knowledgeable in the following areas:

1. Basic electrical and electronic skills, including preparing and following an equipment wiring diagram or schematic.

2. The basics of motion control system applications, such as torque, speed, move distances, how to structure a motion task into move segments and input/output control.

3. Some familiarity with elementary computer programming, including defining the problem to be solved and coding it in a computer language.

1.4 - CONVENTIONS USED IN THIS MANUAL

1. Motor rotation direction (CW and CCW) is properly oriented when viewing the motor from the end opposite the mounting flange end of motor.

2. Please refer to Section 9 “Glossary” for detailed descriptions of terms such as "sink and source I/O", various motion terms, etc.

1.5 - HOW TO CONTACT US

Although this manual represents a detailed compilation of information regarding your SLO-SYN WARPDRIVE control product, sometimes questions may arise which will require that you contact us. You now have a few options available to you when you need information regarding your product or its application:

1. **On the Internet www.danahermotion.com.** Our multimedia enabled web site offers you information such as:

- Free Software
- Software Updates
- TechFax fax on demand documents (1-800-234-3369)
- HTML Product Selector
- HTML Brand Selector
- Product News
- Links
- Sales and Distribution Information
- Product information and specifications
- Literature Requests
- Technical Support E-mail
- Many more features

2. **By Phone**. You may reach us by phoning our Motion Control Applications Engineering Department at telephone (800) 787-3532. We may be reached between the hours of 8:00 am and 5:00 PM (Eastern Time), Monday through Friday. Technical personnel are available to assist you in getting your application up and running as efficiently as possible.

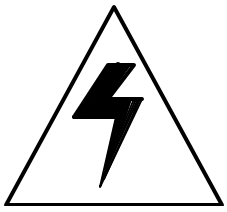
(This page intentionally left blank)

Section 2

Important Safety Information

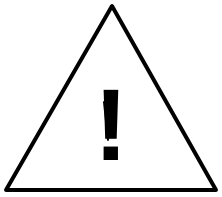
Before installing and operating your Slo-Syn motion control product, it is extremely important both to you and us here at Superior Electric that you read this section very thoroughly and carefully. Your Slo-Syn product will deliver years of reliable, trouble-free, and most importantly, safe operation if you heed the cautions and warnings outlined in this section, and follow the subsequent instructions in the remainder of this manual.

Throughout this manual two very important symbols will be used to identify hazardous and potentially dangerous situations. The symbols are the electrical shock indicator and the exclamation point. Both are always surrounded by a triangle as shown.



Warning

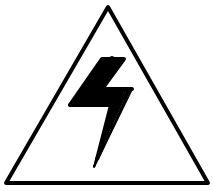
The electrical shock symbol shown to the left is used to indicate situations where ELECTRICAL SHOCK hazards may exist. These warnings must be followed to ensure that YOU avoid electrocution which could result in serious injury or death.



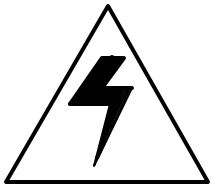
Caution

The exclamation point symbol shown to the left is used to indicate situations other than electrical hazards which may be potentially dangerous to either YOU or to the product. Follow these warnings carefully to avoid injury to you and damage to the product.

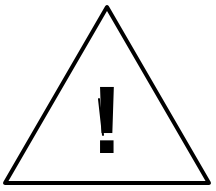
The following indicates a partial list of precautions which must be followed to ensure safe operation of your SLO-SYN unit. Other more specific precautions are indicated in the appropriate sections of this manual. As you read through the manual, pay particularly close attention to these cautions and warnings as they could **save your life!**



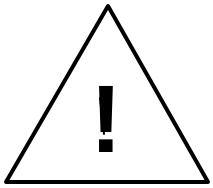
Dangerous voltages, currents, temperatures, and energy levels exist within this unit, on certain accessible terminals, and at the motor. NEVER operate the unit with its protective cover removed! Caution should be exercised when installing and applying this product. Only qualified personnel should attempt to install and/or operate this product. It is essential that proper electrical practices, applicable electrical codes and the contents of this manual be strictly followed .



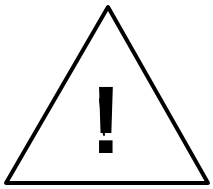
Dangerous high voltages exist in this product. Be certain the power has been removed for a minimum of 5 minutes before any service work or circuit board configuration changes are performed.



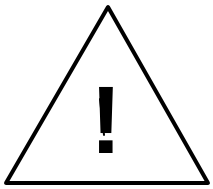
In order to provide the correct levels of protection in the unit, replacement fuses must be the same exact style and ratings as those originally installed in the unit.



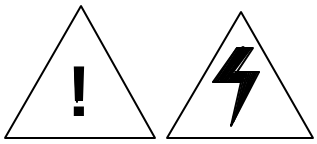
Temperature of the heatsink or the unit could be hot to the touch. Caution should be used when determining the temperature.



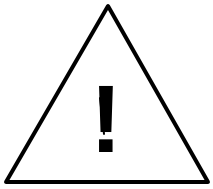
Secure mounting and proper grounding of the Slo-Syn controller/drive is essential for proper operation of the system.



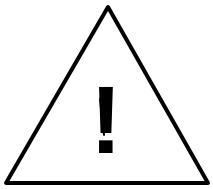
It is your responsibility to follow the appropriate federal, state, and local electrical and occupational safety codes in the application of this product.



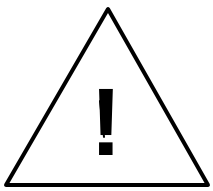
NEVER wire the unit with the power on ! Serious injury as well as damage to the unit may result.



NONE of the inputs to the unit are to be used as an EMERGENCY STOP in ANY application. Although activation of certain inputs will discontinue motion or disable motor current, these are NOT designed as fail-safe E-STOP inputs. Relying exclusively on inputs to the unit to cease motion, which could cause dangerous conditions, is a violation of Machine Safety Codes (ref. IEC 204-1). Other measures such as mechanical stops and fail-safe brakes must be used in these situations.



Step motors can develop high torque and speed. Use extreme caution during development of applications and integration into your system. Sudden motor motion may occur during execution of software programs. All software should be verified for proper operation before integration into your system. The motor may continue to rotate upon removal of power to the unit. It is your responsibility to ensure that no dangerous motion occurs due to gravity loading or free-running motors upon unit shutdown. Fail-safe brakes may be interfaced to the unit to prevent such dangerous conditions.



Step motors can reach surface temperatures up to or exceeding 100 C. Use caution when handling the motors.

(This page intentionally left blank)

Section 3

Quick Start Installation Guide

3.1 - Step-by-Step Start-Up Procedure

The SS2000PCi step motor controller is a sophisticated and versatile product. Setting up the system, however, can be simple and straight-forward if the proper steps are followed. Please use the step-by-step set up guide below.

1) Bench Set Up.

Before connecting your SS2000PCi and drive to your motor and mechanical system or machine, we recommend that you "bench test" the system. This will allow you to become familiar with the wiring, programming and operation of the system before installing it into your machine. This may also prevent inadvertent damage to your mechanical system if you make programming errors which cause unexpected motion. The bench set up can be used to perform simple motions with an unloaded motor. To perform a bench test, do the following:

a) **Wire it up**. Read Section 3.5 Wiring Diagrams, and connect the AC power, I/O and other required signals per the wiring diagrams and instructions. **BE SAFE!!** Do not apply AC power to the unit until you are sure of all connections. Initially, there is no need to connect all of the wiring of your system together. Wire the AC line input, motor drive and HOST communication ports. This will be all you need to establish communications to the unit and perform simple motion.

b) **Load Software.** You will need to use a PC to program the unit according to your requirements. First you must load the MCPI software onto the PC from the floppy disks provided with your unit. Simply insert disk #1 and run the file SETUP.EXE. Once the software is loaded, run it by double clicking on the MCPI icon. See Section 5 for more details on the MCPI installation process.

c) **Create your Project**. You can now create your new **Project.** Your Project will contain **Configuration** information for your particular system, and also your program **Task** which holds the user program written in BASIC-like language. Read section 5 of this manual, and then step through the Configuration folders and enter the appropriate data for your system, saving the configuration when you are done. Note that for this exercise, the original default settings should work fine. Don't forget to set up the serial port for your PC to the correct port number and baud rate.

HINT: Motion is commanded in **User Units.** The System folder in the Configuration allows you to enter User Units per motor revolution. Initially, it is easiest to set this to 1. This will mean that move distances are in motor revolutions (e.g. movei=1 moves one revolution), speeds will be in revs/sec, and accelerations will be in revs/sec/sec. Later this can be changed (e.g. to allow programming in inches on a lead screw) to allow ease of programming once the motor is installed into the mechanical system. See the System Folder section of this manual for other examples. **All** move distances, speeds, and accelerations (or decelerations), and encoder information are provided in User Units, so be sure you understand this before continuing.

d) **Compile and Download** the project into the unit using the command buttons of the MCPI. Note that initially, you can leave the Task blank and command motion using the **Host Commands.** Host commands are entered in **Terminal Mode** from the MCPI. Enter the terminal mode using the appropriate command button on you screen.

e) Make it move! Now that you have compiled and downloaded your project into the unit, you are ready to make the motor move. First you must enter the speed at which you wish the motor to turn, such as 1 rev/sec. Do this by typing speed=1 <CR> (the <CR> means the Return or Enter key). Now enter the acceleration, for example 50 revs/sec/sec by typing accel=50<CR>. Set the deceleration to match by typing decel=50<CR>. After each entry, the controller should respond with a ">" prompt indicating that it has accepted your command. With the motor secured to the bench, you can now command a move. To command an incremental move of 10 revolutions type movei=10<CR>. The motor should now move 10 revolutions. If it does not, check your wiring and verify your configuration settings. In addition, check the motor direction to insure it meets your requirements. The motor direction can be reversed in the System folder if necessary.

f) **Write a BASIC Program**. Now that you have made a simple move, you are ready to write your Task in the MCPI BASIC-like language. Refer to section 6 for a complete description of all of the **Program Commands.** You can start by opening your Task and entering the commands. First, let's enter the exact same commands that you used in the Terminal HOST mode. Enter the speed, accel, decel, and movei commands as you did in step e) above. You must enter two more commands to tell the unit that the program is done after it performs the move. Type waitdone<CR> and End<CR> as the last lines of the program. Since your program has changed, you must compile and download it into the unit again for the changes to take effect. If you receive compilation errors, check your spelling and syntax with the information in section 6.

g) **Execute the Program**. . From the Terminal screen, click on the RUN button to make the motor move 10 revolutions. If desired you can now add lines to the program to perform more sophisticated motion. For example, try typing REAL x <CR> as the first line of your program. This will *declare* **x** as a **REAL variable**. See sections 5 and 6 for discussions regarding variables. On the next line, type x=10 <CR>. This assigns the REAL variable x a value of 10. Change the movei=10 line to movei=x. Now the motor will move whatever distance has been assigned to x. Recompile

and download your program, then run it. It should operate the same as before, but now the program is now using x as the move distance in place of 10 as before. Change the value of x to different distance values to verify that it works correctly.

h) **Expand the Program and Debug it**. Now that you have written a simple program, you can add more complexity by adding more commands. You can do complex looping, access I/O, and motion functions as required. It will be helpful now to use the **DEBUG** feature of the MCPI. Again, refer to section 5 for a detailed description of the debug mode. If you compile your program in Debug Mode, you can enter the debug screen as your program runs and step through your code to verify proper operation. Once the code is functioning correctly, you should re-compile in Release Mode as this will speed up program execution.

2) Installation into Mechanical System

Once you have tested everything out in a controlled environment, you may complete the installation into your system. This will require making all the necessary wiring connections for limit switches, additional I/O, analog inputs, encoder, etc. **Start simple!!** Just as you started with a simple move on the bench, you should start simple here as well, slowly adding complexity as you debug your code and gain more confidence in programming. You may use the Debug Mode to help in this process. Once you have the program running the way you want, you can disconnect the HOST computer and use the RUN switch input or Program Autostart feature in the Configuration to run your program without a computer attached.

3.1.1 - Switch Settings

Caution

Before mounting and wiring your Slo-Syn Controller, the switches that govern various operating features should be checked or set to their proper positions for your application.

Warning

NEVER change the switch settings with the unit powered ON. Risk of physical injury or damage to the unit may result.

3.1.2 - Baud Rate and Unit ID Switch

The Baud Rate switch is accessible through the top of the unit and has two positions, 9600 or User Baud. According to the switch position, upon unit power up or RESET, the baud rate is set to either 9600 or the User Baud rate. If the switch is in the User position, the unit baud rate is set to the baud rate parameter defined in the downloaded project. If the switch is in the 9600 position, the baud rate will be forced to 9600 **regardless of the project configuration**.

It is possible to communicate to multiple units over the same RS-485 transmission lines. To accomplish this, the SS2000PCi supports daisy chain wiring of from 2 to 32 units. **All units MUST have their HOST communications port set to RS-485 mode for daisy chaining to function properly. Insure that the power is off when changing the switch position**. To change the Host port communications mode, slide the RS-232/RS-485 selector switch to the appropriate location. The switch is accessible through the top of the unit near the BCD I/O port. All units must also be set to the same baud rate.

Further wiring details are included in Section 3.5 Wiring Diagrams. Note that **RS-232 daisy chaining is NOT supported**, and RS-232 signals should NOT be connected to the Host port when it is in RS-485 mode.

The Host command <nn allows different modes for daisy chain communications. Refer to the Host Command section for a detailed description of the daisy chain commands including their syntax and usage.

Each unit on the daisy chain must have a unique identification number (ID) to eliminate transmitter conflicts on the RS485 port. Five dip switches are provided for selecting the unit ID (1 – 32). They are accessible through the top rear of the unit. One and only one unit MUST have ID 1. The switch positions are only decoded at power-up. Do not change the switches with the power on.

The unit ID 's are decoded as follows:

ID Num.	SW-1	SW-2	SW-3	SW-4	SW-5
1	ON	ON	ON	ON	ON
2	ON	ON	ON	ON	OFF
3	ON	ON	ON	OFF	ON
4	ON	ON	ON	OFF	OFF
5	ON	ON	OFF	ON	ON
6	ON	ON	OFF	ON	OFF
7	ON	ON	OFF	OFF	ON
8	ON	ON	OFF	OFF	OFF
9	ON	OFF	ON	ON	ON
10	ON	OFF	ON	ON	OFF
11	ON	OFF	ON	OFF	ON
12	ON	OFF	ON	OFF	OFF
13	ON	OFF	OFF	ON	ON
14	ON	OFF	OFF	ON	OFF
15	ON	OFF	OFF	OFF	ON
16	ON	OFF	OFF	OFF	OFF
17	OFF	ON	ON	ON	ON
18	OFF	ON	ON	ON	OFF
19	OFF	ON	ON	OFF	ON
20	OFF	ON	ON	OFF	OFF
21	OFF	ON	OFF	ON	ON
22	OFF	ON	OFF	ON	OFF
23	OFF	ON	OFF	OFF	ON
24	OFF	ON	OFF	OFF	OFF
25	OFF	OFF	ON	ON	ON
26	OFF	OFF	ON	ON	OFF
27	OFF	OFF	ON	OFF	ON
28	OFF	OFF	ON	OFF	OFF
29	OFF	OFF	OFF	ON	ON
30	OFF	OFF	OFF	ON	OFF
31	OFF	OFF	OFF	OFF	ON
32	OFF	OFF	OFF	OFF	OFF

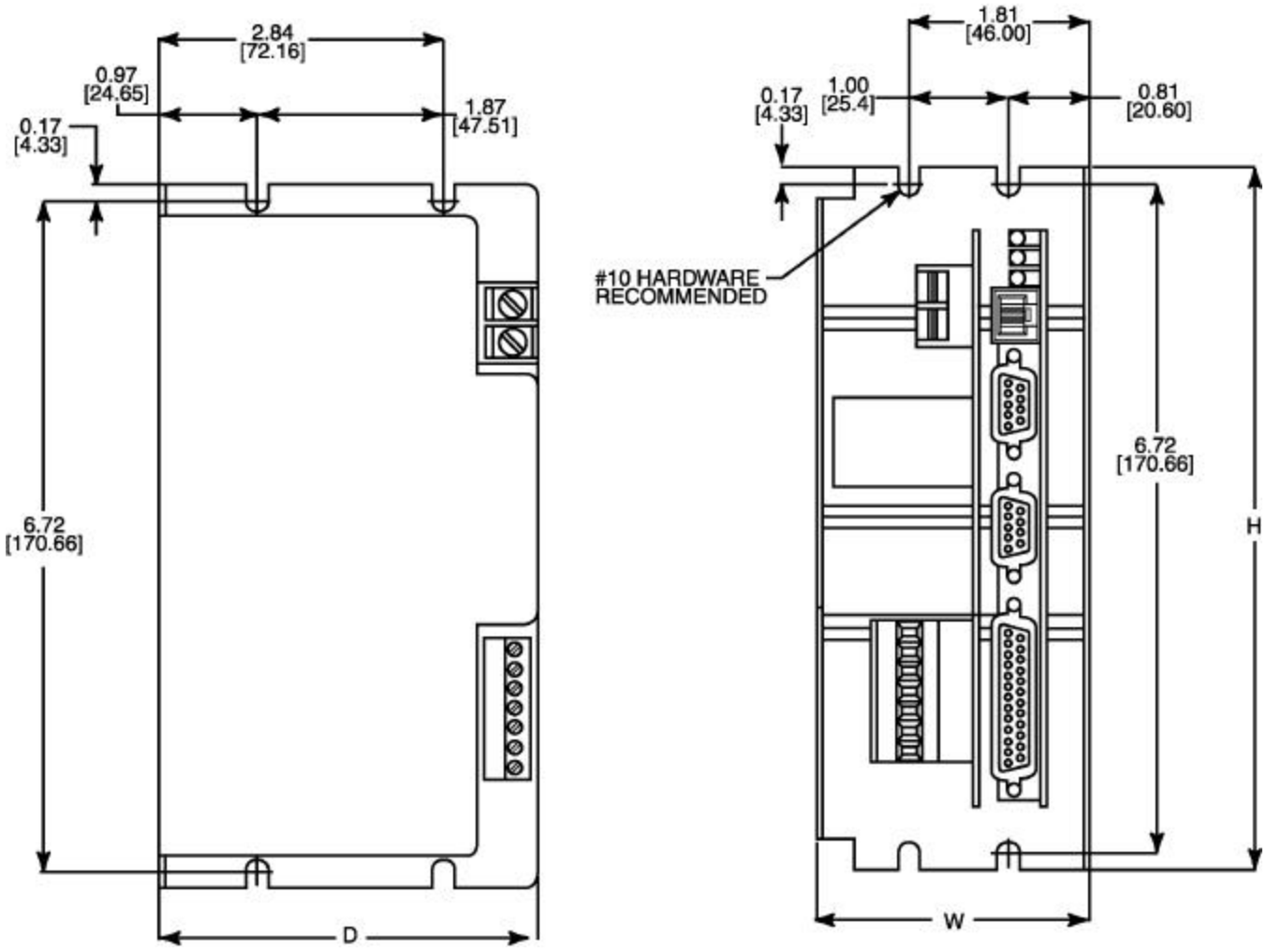


Figure 3.1
Mechanical Outline Drawing

3.2 - Mechanical Mounting of the Unit

Figure 3.1, Mechanical Outline Drawing, provides overall and mounting dimensions for the SS2000PCi. The unit should be solidly mounted within a control enclosure approved for the particular application. It is important to select a mounting location which will meet the environmental specifications listed in Section 4.1 Mechanical and Environmental Specifications. Avoid locations that expose the unit to extremes of temperature, humidity, dirt/dust, or vibration.

At least 2 inches of space must be left on the sides, top, and bottom of the unit to allow proper air flow for cooling of the unit.

Care must also be taken to allow proper and safe access to all wiring. It is best to avoid areas with high electrical noise. As discussed in Section 3.3 General Wiring Guidelines, this will help prevent incorrect operation due to electromagnetic interference.

3.3 - General Wiring Guidelines

Warning

Dangerous voltages, currents, temperatures, and energy levels exist within this unit, on certain accessible terminals, and at the motor. NEVER operate the unit with its protective cover removed! Caution should be exercised when installing and applying this product. Only qualified personnel should attempt to install and/or operate this product. It is essential that proper electrical practices, applicable electrical codes and the contents of this manual be followed strictly.

Superior Electric SLO-SYN controls and drives use modern solid-state digital electronics to provide the features needed for advanced motion control applications. Although care has been taken to ensure proper operation under a wide range of conditions, some user equipment may produce considerable electromagnetic interference (EMI) which can cause inappropriate operation of the digital logic used in the control, drive, or other computer-type equipment in the user's system.

In general, any equipment that causes arcs or sparks or that switches voltage or current at high frequencies can cause interference. In addition, ac utility lines are often "polluted" with electrical noise from sources outside a user's control (such as equipment in the factory next door). Some of the more common causes of electrical interference are:

- power from the utility ac line
- relays, contactors and solenoids
- light dimmers
- arc welders
- motors and motor starters
- induction heaters
- radio controls or transmitters
- switch-mode power supplies
- computer-based equipment
- high frequency lighting equipment
- dc servo and stepper motors and drives

The following wiring practices should be used to reduce noise interference.

Solid grounding of the system is essential. Be sure that there is a solid connection to the ac system protective earth ground (PE). Insure that there is a good electrical connection through the controller case to the control system enclosure . A separate grounding strap may be required to properly ground the unit to the control system enclosure. This strap should ideally be constructed using copper braid at least 0.5" in width. Use a single-point grounding system for all related components of the system (a "hub and spokes" arrangement). Keep the ground connection short and direct. Grounding through both a mechanical connection to the control enclosure and through a grounding strap is optimal.

Keep power and signal wiring separated. Power wiring includes ac wiring, motor wires, etc. Signal wiring includes inputs and outputs (I/O), encoder wiring, serial communications (RS232 lines), etc. If possible, use separate conduit or ducts for each. If the wires must cross, they should do so at right angles to minimize coupling.

Use separately bundled, shielded, twisted-pair cables for the drive to motor, encoder, serial communications, analog input, and digital I/O wiring. For other connections it is recommended that the shields be terminated at the Slo-Syn unit as well. Shield connections are provided on the unit terminal connectors for this purpose. All cable shielding should be terminated at ONE END ONLY. Grounding the serial communications connections at the opposite end from the controller may be necessary in some systems. If the cable shield must be connected at the opposite end from the Slo-Syn unit, the shield should NOT also be connected at the unit as this may cause a "ground loop" and introduce electrical noise problems.

Suppress all relays as close to the coil as possible to prevent noise generation. Typical suppressors are diodes, capacitors or MOV's. (See manufacturer's literature for complete information). Whenever possible, use solid-state relays instead of mechanical contact types to minimize noise generation.

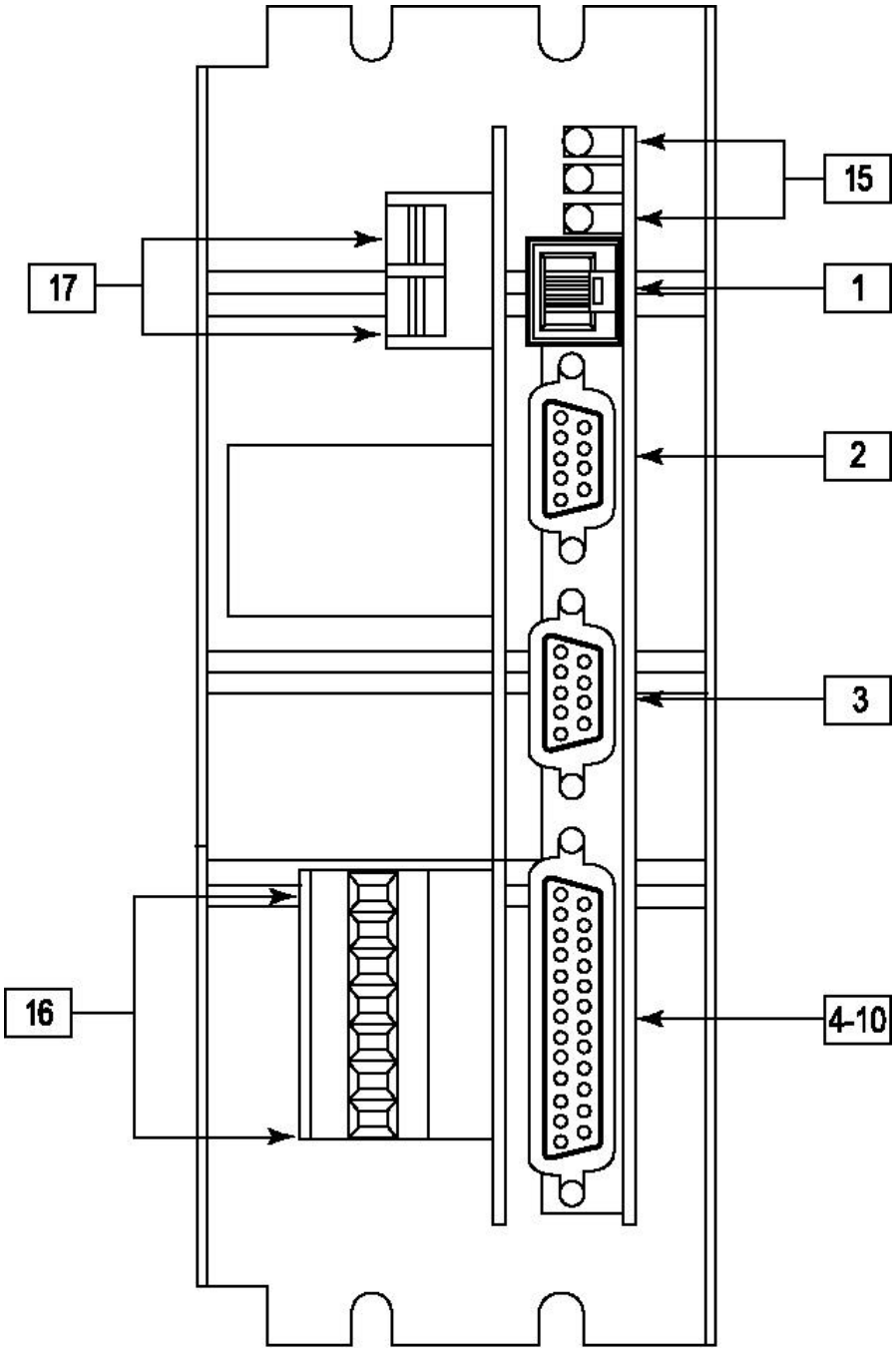
In some extreme cases of interference, it may be necessary to add external filtering to the ac line(s) feeding affected equipment, or to use isolation transformers to supply their ac power.

NOTE: Superior Electric makes a wide range of ac power line conditioners that can help solve electrical interference problems. Contact 1-860-787-3532 for further assistance.

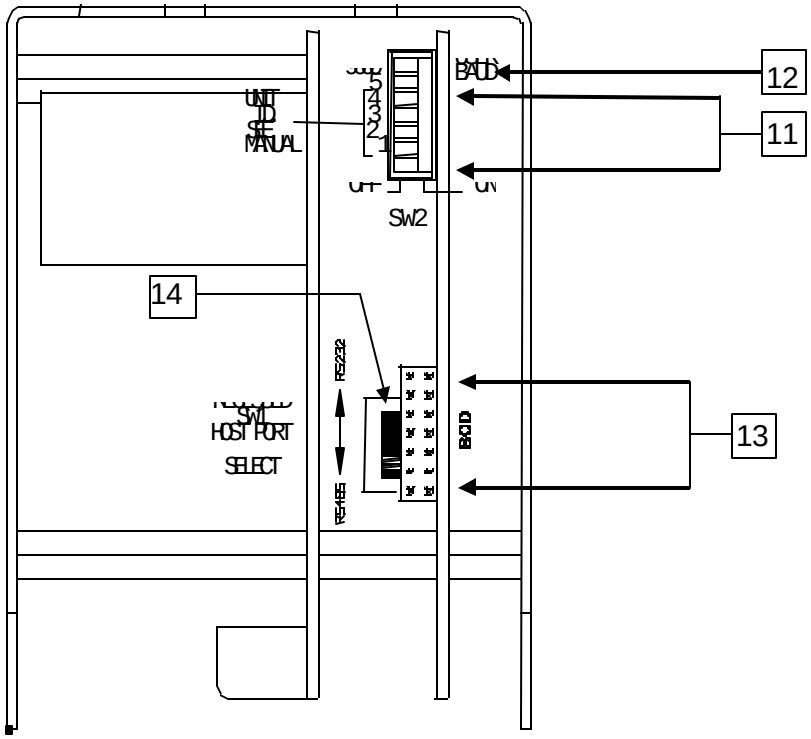
3.4 - Hardware Connection Descriptions

The following figures indicate the side, top, and front views of the SS2000PCi controller. The numbers in the boxes show the position of the various hardware connections to the unit. Use the index number in the boxes to find the description of each connection following the diagrams. The descriptions given here should provide a reasonable understanding of the nature of each signal and the way it should be wired into your system. More detailed technical information is available in Section 4.0 Hardware Specifications.

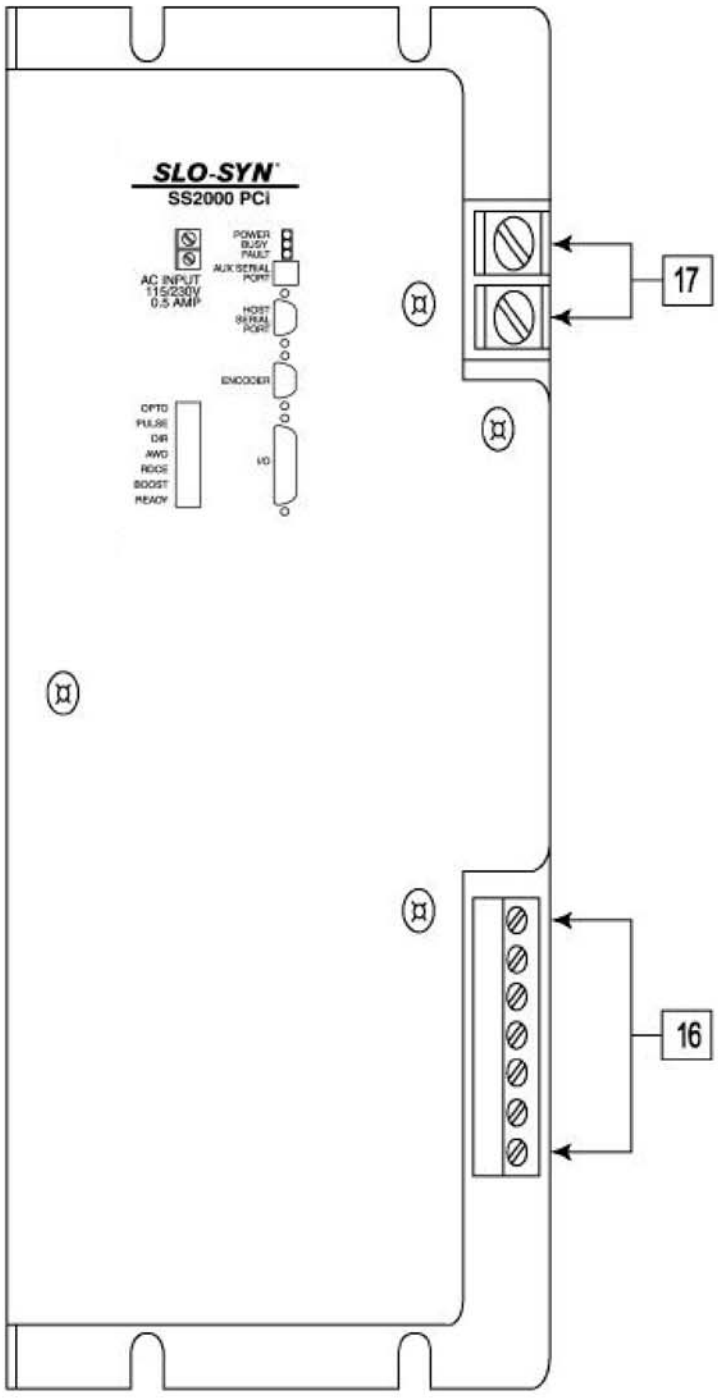
FRONT VIEW



TOP VIEW



SIDE VIEW


SLO-SYN
SS2000 PCi
AC INPUT
115/230V
0.5 AMP
POWER
BUSY
FAULT
AUX SERIAL PORT
HOST SERIAL PORT
ENCODER
I/O
OPTO
PULSE
DIR
AWO
RDCE
BOOST
READY
17
16

The following are brief descriptions of each connection to the D6i unit. More detailed wiring diagrams are provided in Section 3.5

1 Serial Port 2

Auxiliary Serial Communications: Port 2

TX2+: Transmit+ for Serial Port 2 (RS485)
TX2-: Transmit- for Serial Port 2 (RS485)
RX2-: Receive- for Serial Port 2 (RS485)
RX2+: Receive+ for Serial Port 2 (RS485)

2 Serial Port 1

Host Serial Communications: Port 1

TX1-: Transmit- for Serial Port 1 (RS485)
TX1+: Transmit+ for Serial Port 1 (RS485 / RS232)
RX1+: Receive+ for Serial Port 1 (RS485 / RS232)
RX1-: Receive- for Serial Port 1 (RS485)
Serial GND: Signal ground for Serial Port 1
+5V: **DO NOT USE AT THIS TIME**.

3 Encoder

Encoder inputs for a closed loop stepper can be single-ended or differential phase quadrature.

B1+: Encoder Channel B+ input
B1-: Encoder Channel B- input.
A1+: Encoder Channel A+ input.
A1-: Encoder Channel A- input.
Z1+: Encoder INDEX Channel Z+ input.
Z1-: Encoder INDEX Channel Z- input.
+5V: +5V supply for encoder.
GND: Ground for encoder.

4 Inputs 1-8 (Isolated)

EVENT 1/ IN1; EVENT 2 / IN 2

These inputs can be used as mark registration and/or home inputs. If the inputs are not used for mark registration or home then the inputs can be used as programmable inputs. These inputs can be configured in the *Project Configuration & Setup*.

+LIMIT / IN3; -LIMIT / IN4

The +LIMIT or the -LIMIT may be used as inputs for limit switches or sensors. If limit switches are not needed, the inputs can be configured in the *Project Configuration and Setup* as programmable inputs.

RUN / IN5

The run input will start execution of the program. If auto-start is selected the program will start upon power up or RESET. RUN will also re-start a program if a **CLEAR** has been activated, or resume a program if a **FEED-HOLD** has been activated. If the **RUN** input is not needed the input can be used for a programmable input. This is done in the *Project Configuration & Setup.*

CLEAR / IN6

If the **CLEAR** input is open, the program or motion will stop. This input must be closed to run the program or start motion. If the **CLEAR** input is not needed the input can be used for a programmable input. This input can be configured in the *Project Configuration & Setup.*

FEEDHOLD / IN7

Activation of this input will cause motion to come to a **controlled stop**. After release of the **FEEDHOLD** input, activation of the **RUN** input will continue the program from the point the **FEEDHOLD** was activated. If the **FEEDHOLD** input is not needed it can be used as a programmable input. This input can be configured in the *Project Configuration & Setup.*

IN 8

This input can be used as a programmable input.

5 Outputs 1-4 (Isolated)

OUT 1, OUT 2, OUT3, OUT 4

These outputs can be used as programmable outputs.

6 Analog Input

The analog input connection allows a voltage from 0 VDC to +10VDC to be read into the unit.

ANALOG IN: analog input.
GND: Ground for analog input.

7 OPTO

+VOPTO; -VOPTO

A power supply for the optical isolators is **REQUIRED** *for proper I/O operation. This supply must be connected to the +VOPTO and -VOPTO pins.* The +12VDC and +12V COM power supply is available. This supply **MUST** be connected to +Vopto and -Vopto unless the user is to supply power for the I/O from a different source.

8 12 VDC Power Supply

12 VDC is available to power I/O . It is recommended that this power be connected to the +Vopto and -Vopto inputs on the controller as the discrete I/O supply.

+12V: +12VDC output.
12V COM: Common for the 12VDC supply.

The +12VDC supply current is limited to 100 mA. See Section 3.5.2 for connection of sink or source I/O.

9 IN COMMON

This input determines the current source of Inputs 1-8. If it is connected to +Vopto the inputs are set to the sinking mode. If it is connected to -Vopto the inputs are set to the sourcing mode.

10 GND

Signal Common for the **Analog In** signals. **This GND is not connected to 12VCOM or IN COMMON**.

11 Device ID Number Switch

The DIP switches will allow up to 32 devices to be daisy chained together. Each unit must have a unique ID number per the table of ID settings in Section 3.2, Baud rate and Unit ID Switches.

12 Baud Rate Switch

This switch is read only at power-up or after a reset command. In the **off** position the baud rate is forced to 9600. In the **on** position the baud rate for the loaded project is used. The User Baud rate is selected in the project *Configuration and Setup*. If no user program is loaded the default, 9600, baud rate is used. If the baud rate in the configuration and setup is not known, use 9600 at power-up.

13 BCD Port

BCD Port / I/O

This port can be used as either a BCD port, consisting of 7 numbers and a sign **(Superior Electric Part # 221157-002, includes BCD switch and 18" ribbon cable)**, or used for additional outputs and inputs*.

BCD0/IN9 : BCD switch data 0 or program input 9.

BCD1/IN10: BCD switch data 1 or program input 10.

BCD2/IN11: BCD switch data 2 or program input 11.

BCD3/IN12: BCD switch data 3 or program input 12.

BCD4/IN13: BCD switch data 4 or program input 13.

BCD5/IN14: BCD switch data 5 or program input 14.

BCD6/IN15: BCD switch data 6 or program input 15.

BCD7/IN16: BCD switch data 7 or program input 16.

BCD STR0/OUT5 : BCD switch Strobe 0 or output 5

BCD STR1/OUT6 : BCD switch Strobe 1 or output 6

BCD STR2/OUT7 : BCD switch Strobe 2 or output 7

BCD STR3/OUT8 : BCD switch Strobe 3 or output 8

GND: Signal Common for Inputs and outputs

***Note: When the BCD port is used for additional I/O, all inputs are non-isolated, and all outputs are open-collector (7406) active low.**

14 Serial Port 1 (RS232 / RS485) Switch

This switch allows Serial Port 1 to be configured for RS232 or RS485 4-wire communications.

Caution

Use care when accessing this recessed switch; do not damage adjacent components when changing its position.

15 LED's

These LED's show conditions that may be occurring in the controller.

POWER: The power LED indicates that there is AC power applied to the drive and that the logic supply is active.

BUSY: Signifies that motion is occurring on the motor.

FAULT: Indicates that an error has occurred in the controller.

16 Drive Connections

- **OPTO** — +5VDC output power for connection to drive opto coupler supply.
- **PULSE**— Open collector output that supplies pulses to the drive. Drive should count ON-OFF transitions (low to high)
- **DIR** — Open collector output that signals drive which direction to turn the motor.
- **AWO** — Open collector output that turns off drive when on (low).
- **REDUCE** — Open collector output that signals drive to reduce motor current when active (low).
- **BOOST** — Open collector output that signals drive to boost motor current when active (low).
- **READY** — Logic input signal indicating that the drive is ready to go. This signal is active when at +5V DC (OPTO power).

17 AC Power

AC CONNECTIONS

These inputs are for connection of single phase AC power. The input power range is from 95VAC to 265VAC. 50/60 HZ.

18 Chassis Ground

Grounding locations for the motor and AC connections. It is critical that a solid connection from Protective Earth Ground be connected to the chassis ground. The Ground wire must be at least as large as the AC supply power wiring.

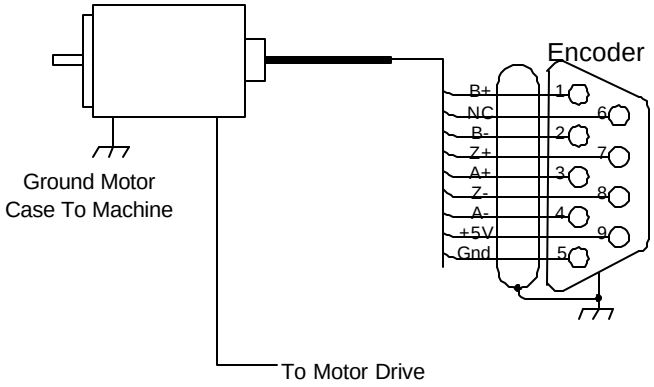
3.5 - Wiring Diagrams

This section provides wiring diagrams for each connection. Remember to follow the General Wiring Guide outlined previously.

NEVER wire the unit with the power on! Serious injury as well as damage to the unit may result.

3.5.1 - Encoder Connections

The encoder connections are only required for a closed loop stepper drive and the connection scheme is depicted below.
Note: It is **IMPORTANT** that the encoder and motor cables be shielded and that the shields be connected to their appropriate connector terminals. Single ended TTL or open collector encoder signals must be connected to the "+" terminal of each channel.

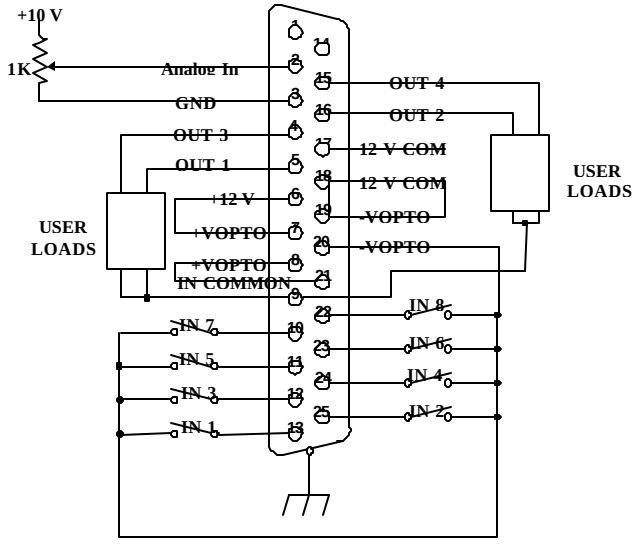


3.5.2 - Input / Output Connections

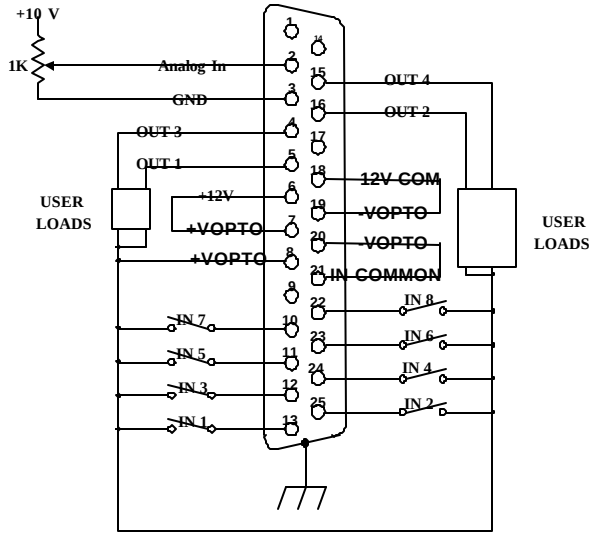
The I/O connections consist of; 8 general purpose inputs, 4 general purpose outputs, and 1 analog input. The 8 general purpose input signals can be sinking or sourcing opto-isolated inputs. The input mode (sink or source) applies to all 8 inputs. They may not be individually selected as sink or source. The 4 general purpose output signals are sinking only opto-isolated outputs. The analog input has a voltage range from 0 to +10 volts.

I/O Connection examples using the +12 Vdc internal power supply are depicted below. The general purpose input connections are shown for both sinking and sourcing inputs.

Internal Power Supply
Inputs Sinking – Outputs Sinking

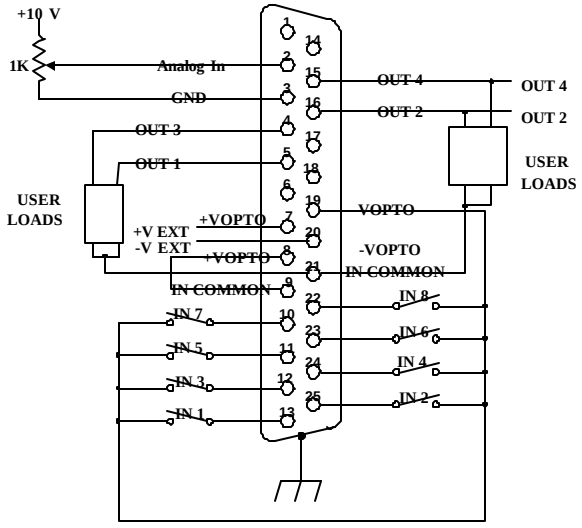


Internal Power Supply
Inputs Sourcing – Outputs Sinking

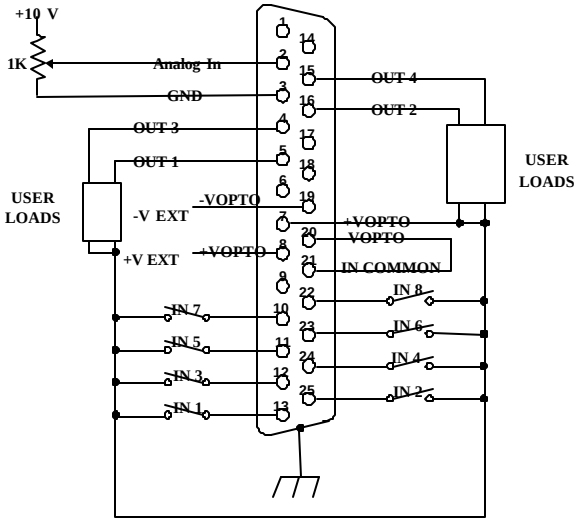


Depicted below are the I/O connections when using an External Power supply. Use these connections if you are not using the +12 Vdc internal power supply. The general purpose input connections are shown for both sinking and sourcing inputs.

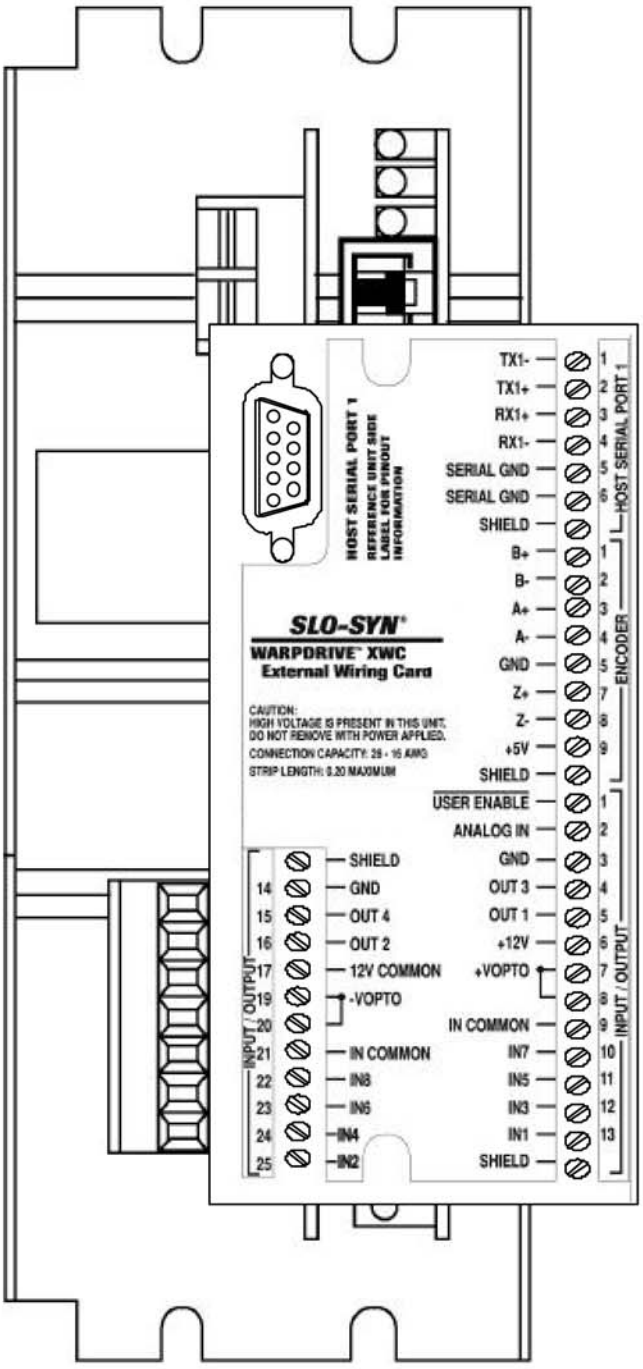
External Power Supply
Inputs Sinking – Outputs Sinking



External Power Supply
Inputs Sourcing – Outputs Sinking

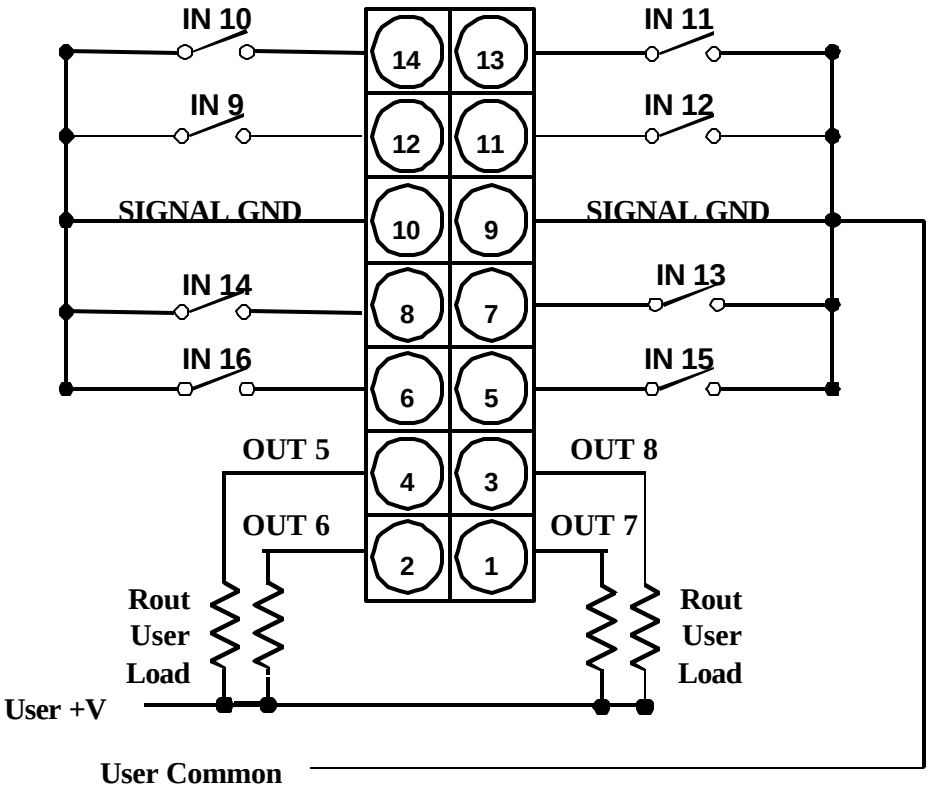


An optional external wiring card is available which provides individual terminals for each I/O point on the "D" style connectors utilized on the unit. The External Wiring Card mounts over the D connectors on the face of the unit. All connections are brought out to individual clamp down type terminal connections. The pinouts for the terminal blocks are identical to the D connectors described in this section. If you would like to utilize the optional external wiring card contact customer service or your distributor and request Part Number XWC. The optional external wiring card is shown below. Note: The USER ENABLE is not used when the XWC is used with the SS2000PCi.



The connections for the BCD I/O connector when used as general purpose signals are depicted below.

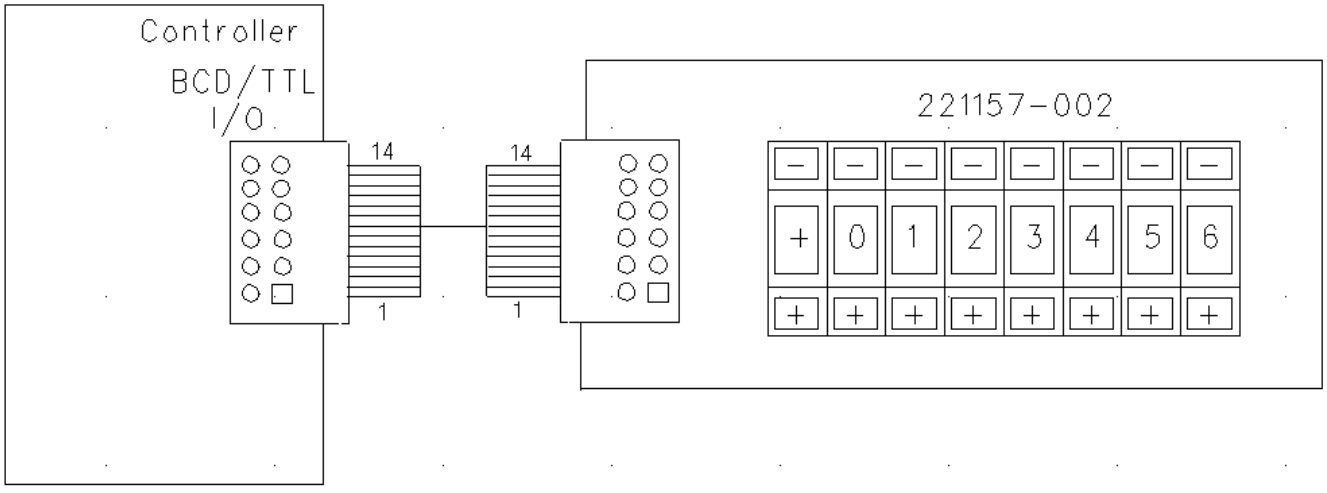
Note: These inputs and outputs are not isolated.



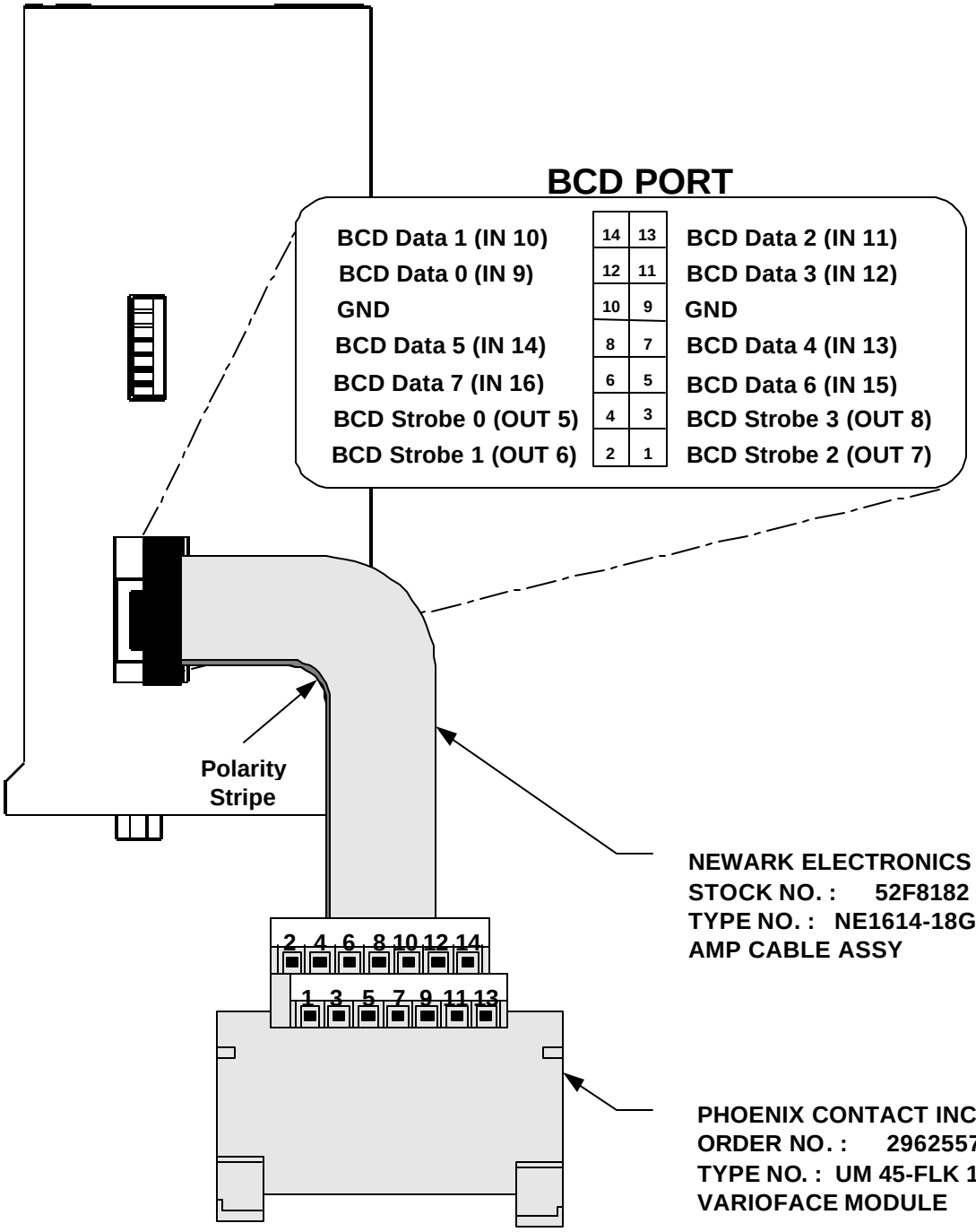
Recommended Output Loads

User + V	**Rout**
5 VDC	500 ohm
12 – 15 VDC	1.5 Kohm
24 VDC	2.5 Kohm

The connections for the BCD / TTL I/O connector when used as a BCD port are depicted below.
Note: The Superior Electric BCD switch interface P/N 221157-002 is shown.

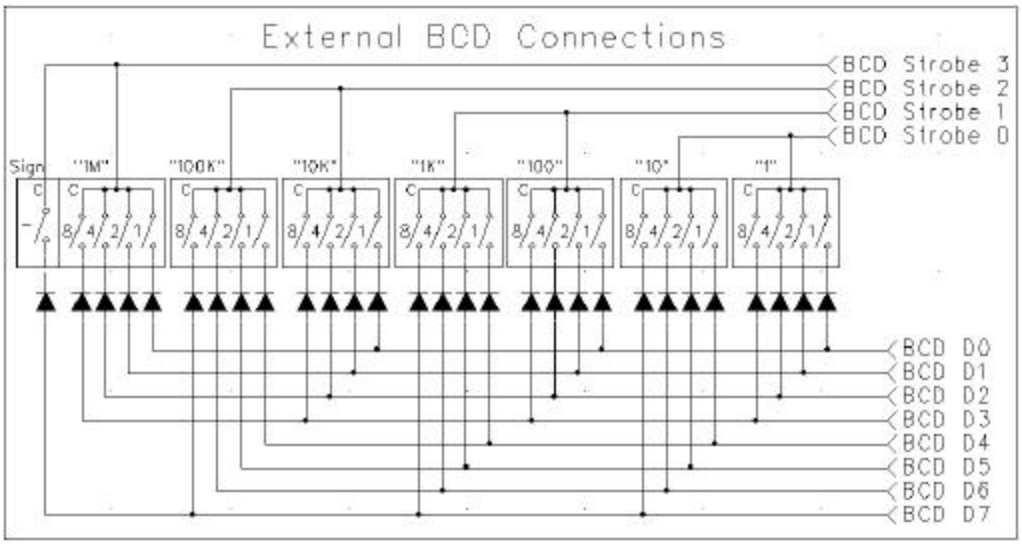


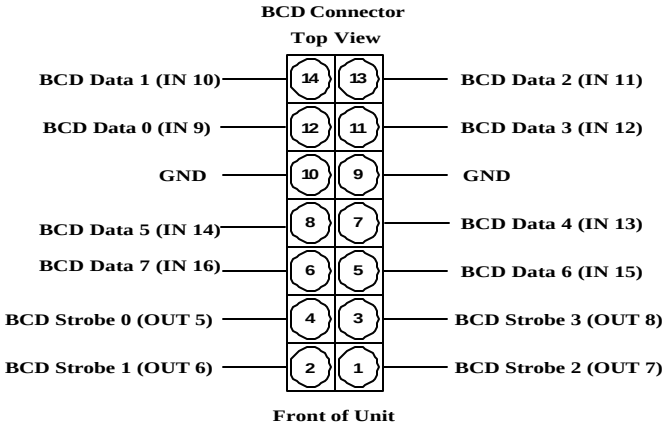
RECOMMENDED CONNECTION FOR USING BCD PORT AS NON-ISOLATED I/O



The connections for the BCD / TTL I/O connector when used as a BCD port are depicted below.

Note: An External BCD Configuration is shown below.

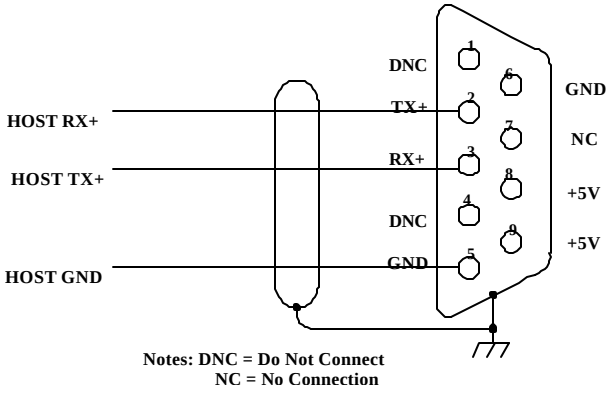




3.5.3 RS232 / RS485 Host Serial Communications Connections

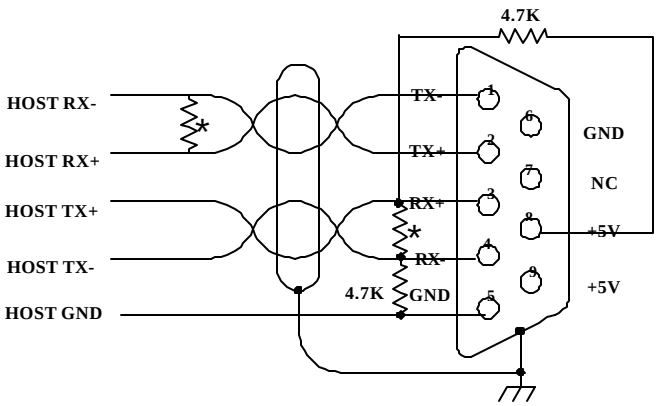
This serial port is used for communications and programming of the controller from a personal computer (PC). The port can be configured for RS232 or RS485 operation. A slide switch has been provided for making this selection. The factory default is RS232. The connection diagram for both modes has been provided, See Section 3.4 Top View. **Note: When wired for RS485 operation, a cable with individual twisted pair wires must be used.** The termination resistors indicated with an * must have a value of 120 ohms. The pull up/down resistors have a value of 4.7kΩ. If multiple units reside on the RS485 bus only ONE set of three resistors should be used at the end of the transmission line bus. The resistor connected to the TX+ and TX- signals may be required if the terminal device does not provide the termination resistor internally. If the terminal device does provide the resistors internally, the resistors connected to the TX+ and TX- are not required.

Host (RS232 Selected)



Notes: DNC = Do Not Connect
NC = No Connection

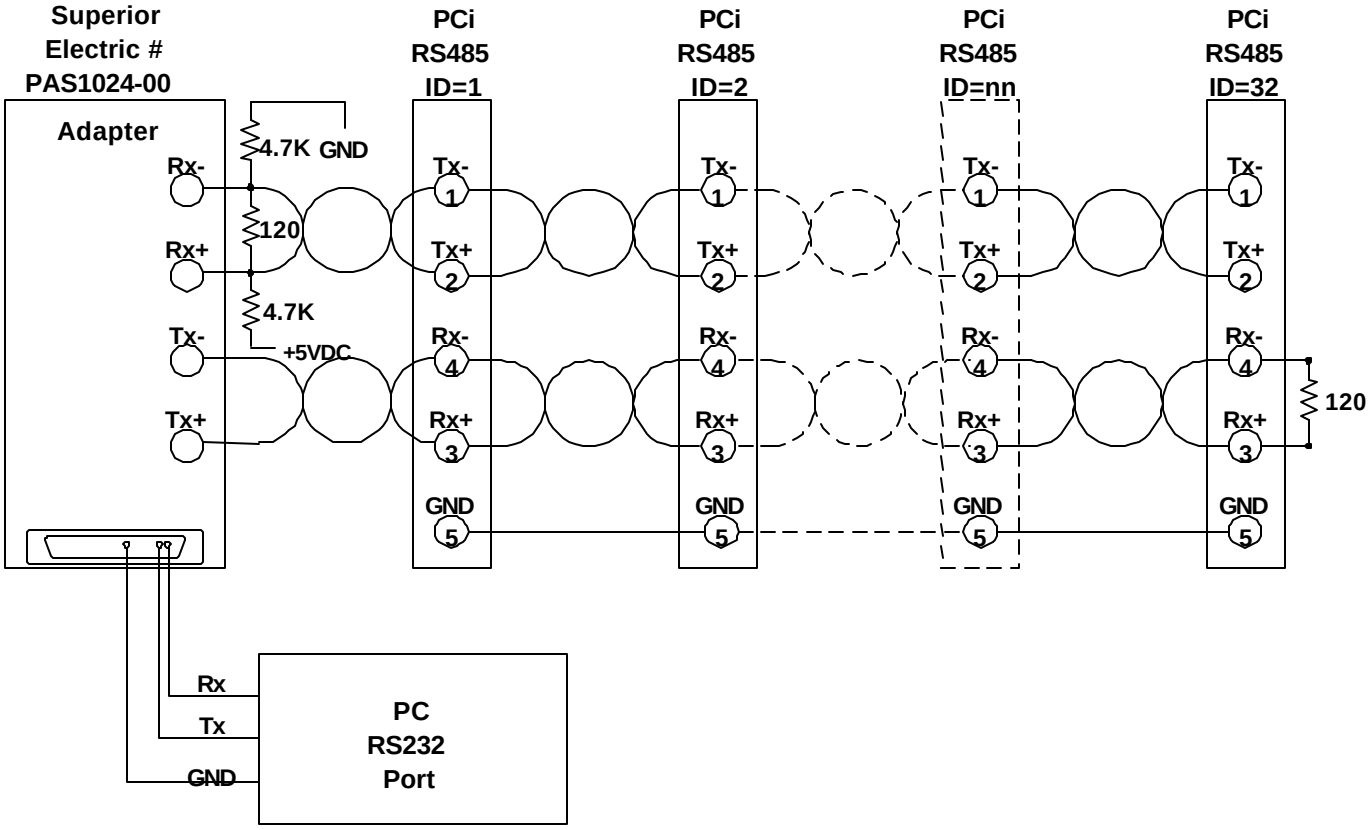
Host (RS485 Selected)



3.5.3.1 - RS485 Host Daisy Chaining Connections

Connection in a daisy chain configuration requires that the Host port of all units be wired as RS-485. Each unit must also be switched to RS485 Host communications mode by moving the recessed slide switch into the RS485 position. The switch is accessible through the removable portion of the label on top of the unit near the BCD I/O port. Be sure that the unit is off when changing the switch position.

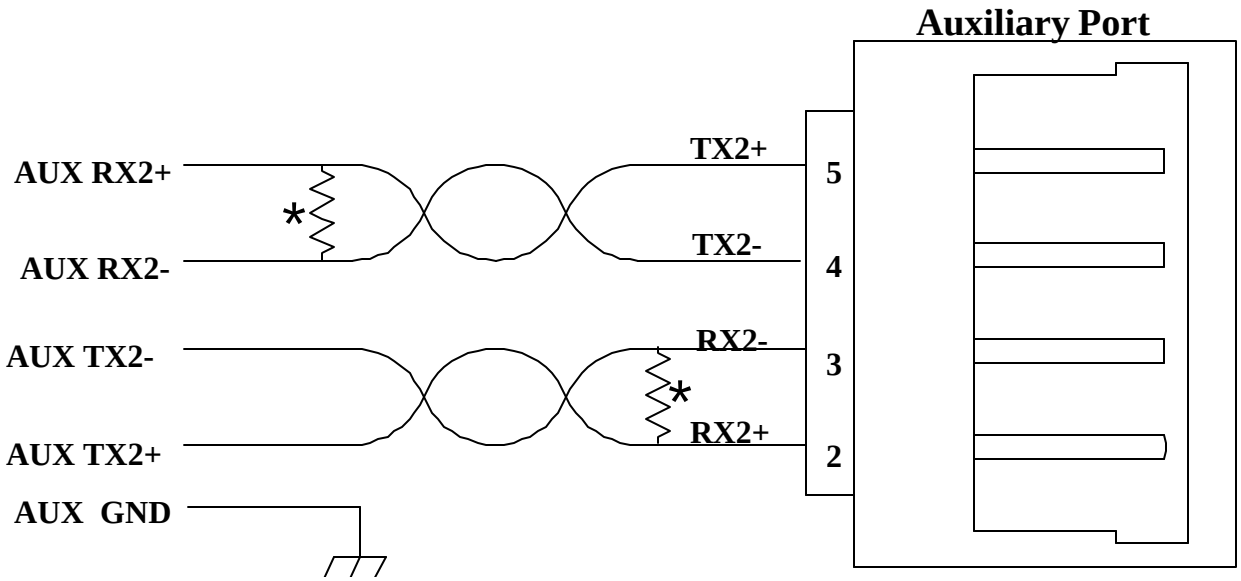
<u>Important!!</u> Connection to a PC that has an RS-232 port only can be accomplished by using an RS-232 to RS-485 four wire adapter such as Superior Electric part number PAS1024-00 as shown. If your PC has an RS-485 port, the adapter is not required.



3.5.4 RS485 Auxiliary Communication Connections

The auxiliary serial port is used for serial communications to and from other devices, such as PLC's or operator interface panels. This serial port is RS485 only and uses a telephone jack for the connections. The wiring connection diagram is shown below. **Note: A cable with individual twisted pair wires should be used.** The termination resistors indicated with an * must have a value of 120 ohms.

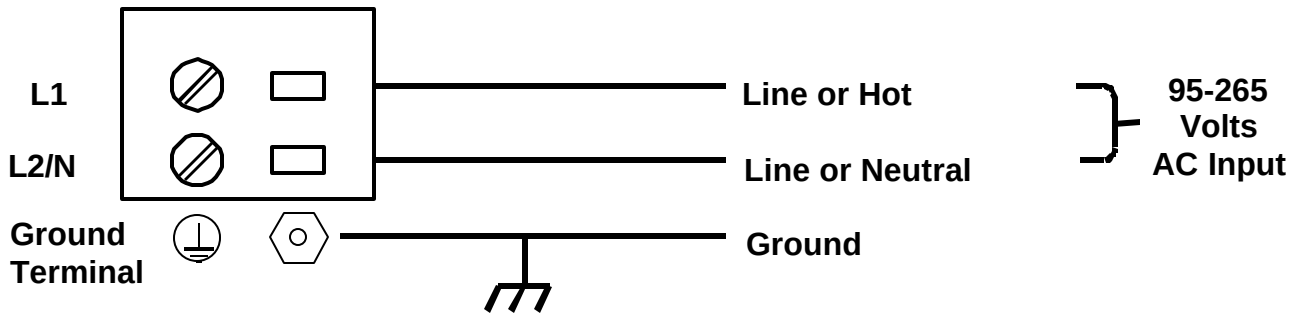
If multiple units reside on the RS485 bus, ONE resistor should be used at the end of the transmission line bus. The resistor across the TX+ and TX- signals may be required if the terminal device does not provide a termination resistor internally. If the terminal device does provide the resistor internally, the resistor across TX+ and TX- is not required.



3.5.5 - AC Power Connections to the Unit

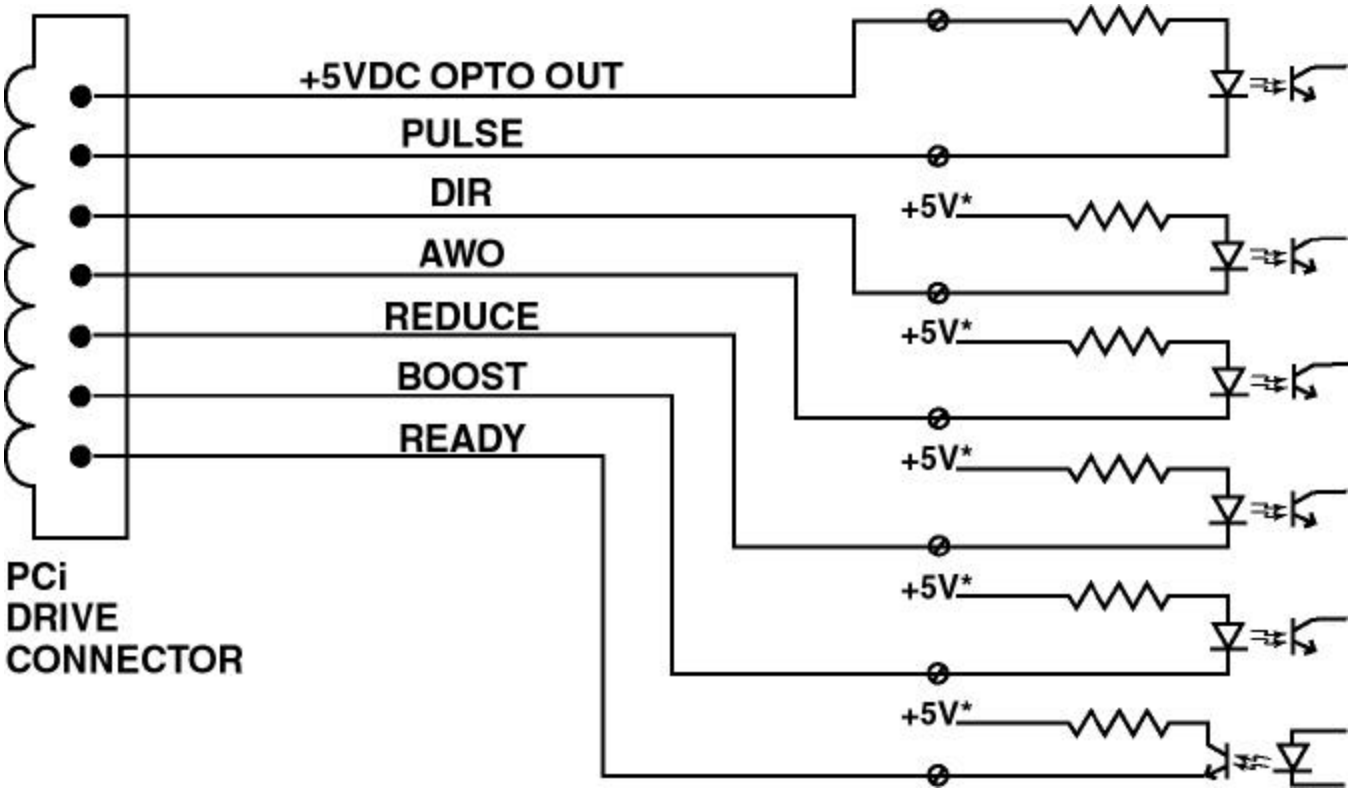
Connect the two (2) AC IN terminals to the input AC line. The line voltage can be from 95 VAC to 265 VAC 50/60 Hz.

Do not exceed the voltage rating of the drive and motor. Damage may occur if the ratings are not observed.



3.5.6 Drive Connections

A typical optically isolated connection to a stepper drive is shown below:



* Note that on some drives the connection from the top side of the opto couplers is made internally. Other drivers may have two terminals per I/O point (uncommitted opto couplers). These drives would require external connection from the high side of the opto coupler to +5VDC opto out. A current limiting resistor may also be required. Consult your drive manual and verify proper connections prior to energizing your system.

(This page intentionally left blank)

Section 4

Hardware Specifications

4.1 Mechanical and Environmental Specifications

Size: (Inches)	2.74W x 7.06H x 3.79D
(mm)	69.57W x 179.32H x 96.2D
Weight	7.75 lbs (3.52 kg)
Operating temperature:	+32º F to +122º F (0º C to +50º C)
Storage temperature:	-40º F to +167º F (-40º C to +75º C)
Humidity:	95% maximum, non-condensing
Altitude:	10,000 feet (3048 meters) maximum

4.2 Electrical Specifications

AC Input Range	95 to 265 VAC, 50/60 Hz
AC Current	.5 amperes
Fuse Rating**	250 volts, 2 amperes
Fuse Type**	Type 3AG

** If this fuse blows, the power supply will be prevented from energizing any of its outputs, hence, the unit will not operate. Usually, this fuse will only blow if an internal failure occurs. In order to ensure safety the specified rating and type of fuse **MUST BE USED**.

4.2.1 Isolated Digital I/O

12 VDC I/O Power: 11.5 to 14 VDC @ 100 mA

Inputs (IN1 – IN8):

Sink mode: (IN COMMON tied to +Vopto)

On state voltage range (+Vopto =12V with -Vopto = 0V):	0V to 6VDC
Input Current; (VIN = 0V), +Vopto=12V, -Vopto=0V:	-6mA

Source mode: (IN COMMON tied to –Vopto = 12V common)

On state voltage range with -Vopto = 0V:	4.5V to 24VDC
Input Current; (VIN=12V) with -Vopto=0V :	6mA

Response time (sink or source):

Opto turn on delay:	10µS typical
Opto turn off delay:	75µS typical

User OPTO coupler power supply range
(if not using the internal +12 V supply): 5-24 VDC

Programmable Outputs (OUT1 – OUT4):

Sink mode only:

Continuous Current rating:	250 mA max
Maximum collector voltage with -Vopto = 0V:	25V
On state voltage @ 250mA:	1.5V max

4.2.2 Non-isolated I/O (or BCD Interface):

IN 9 - IN16:

These inputs may be used with open collector outputs without an external supply by connecting the output device common (ground) to signal ground on the unit, and the open collector to the input pin. An internal pullup resistor to +5VDC is provided

Logic high input level: Open circuit or sourcing voltage	**25V > Vsource > 4.5V, Or Open Circuit**
Logic low input level:	**1.2V max**
Logic low current with input @ GND:	**-1mA max**

OUT 5 – OUT8:

These are open-collector, sink only outputs which are NOT isolated from the unit's +5 V logic supply. Proper care must be exercised to insure noise is not injected onto these signals. The user's I/O supply must be referenced to GND on the controller (e.g. at BCD port pins 9 &10).

Active output voltage:	**.6V max @ 20mA**
Permissible output current:	**20mA**
Permissible output voltage:	**24VDC**

4.2.3 Serial Communications

Port 1:

Configurable for RS-232C or RS-485 four wire specifications via a switch. For RS-232 mode RX1+ is used to receive data into the unit, and TX1+ is used as the transmit data out. Port 1 is designated as the HOST communications port.

In RS-485 four wire mode, with longer distances, the transmission line should be terminated at the end opposite from the source with a 120 ohm termination resistor. Termination resistors are NOT internal to the unit. The 485 transmitter on the unit is tri-stated when transmission is not occurring.

For RS-232:	High level output voltage, VOH:	5Vmin
	Low level output voltage, VOL:	-5Vmax
	Input impedance:	Approx. 3Kohms
For RS-485:	High level output voltage, VOH:	3V min
	Low level output voltage, VOL:	.5V max
	Idle transmission state:	High impedance

Port 2:

Serial channel 2 is RS-485 and is used for differential four wire USER communications. For longer distances, the transmission line should be terminated at the end opposite from the source with a 120 ohm termination resistor. Internal termination resistors are NOT included in the unit. The 485 transmitter on the unit is tri-stated when transmission is not occurring.

High level output voltage, VOH:	3V min
Low level output voltage, VOL:	.5V max
Idle transmission state:	High impedance

4.2.4 Drive Connections

PULSE, DIR, AWO, BOOST, REDUCE: Open collector. Maximum collector voltage +24VDC
Maximum on current: 20 mA
Note: Pulses valid at low to high transition on PULSE output.

READY: Maximum input voltage = DRIVE OPTO OUT (+5VDC)

4.2.5 Encoder Connections

Encoder Connections provide power and inputs for a digital encoder interface to indicate motor position to the controller. Differential connections to the encoder port are highly recommended for noise immunity. If single ended TTL encoders are used, connect them to the "+" terminals of each channel.

Encoder +5VDC power supply output:	+5VDC (+/- 5%) @ 100mA current.
Encoder signal inputs:	TTL level single ended or differential channels A and B in phase quadrature.
Input current A+,A-,B+,B-,Z+,Z-:	+/- 5mA min
Max Input Frequency	500 KHz per channel, 2 megacounts in quadrature

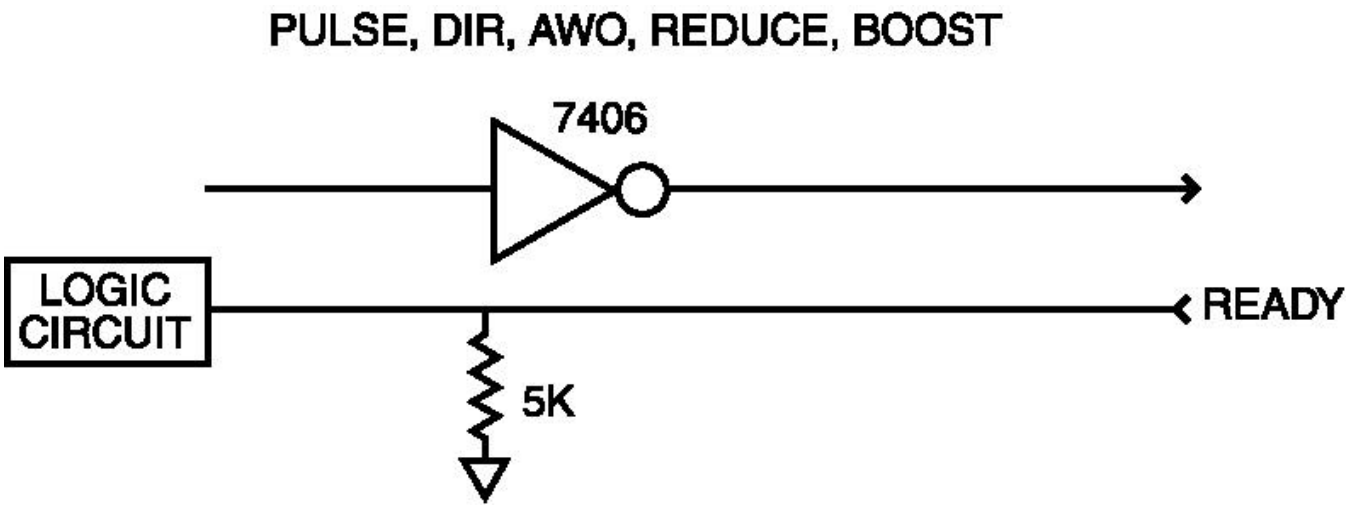
4.2.6 Analog Input

Voltage Range:	+10V(max) to 0V (min) referenced to GND
Resolution:	10 bits or 9.77mV
Absolute Accuracy:	+/- .3V max
Sample Rate:	500 Hz min
Bandwidth:	100 Hz max

4.3 - Hardware Equivalent Circuits

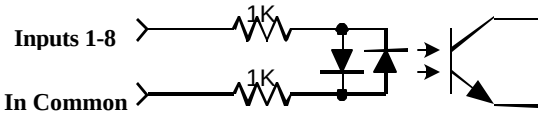
The following pages contain equivalent circuit diagrams which represent the physical interface hardware internal to the unit. These circuits may be used as a reference if questions arise during detailed system design and integration. These diagrams are intended to represent the interface circuitry as accurately as possible, however Superior Electric reserves the right to make minor improvements which may change the exact nature of the circuitry while not degrading functionality.

Drive Connections

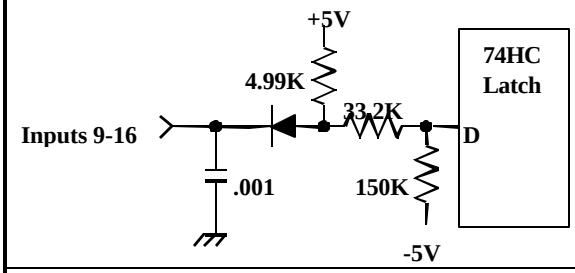


I/O Equivalent Circuits

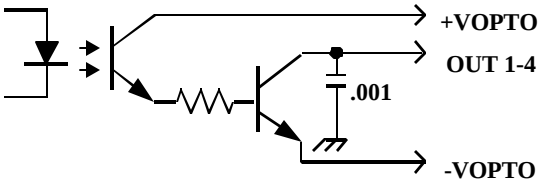
Inputs 1-8 (sink/source)



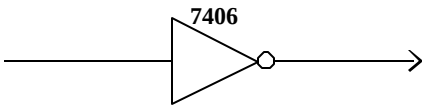
Inputs 9-16 (BCD inputs)



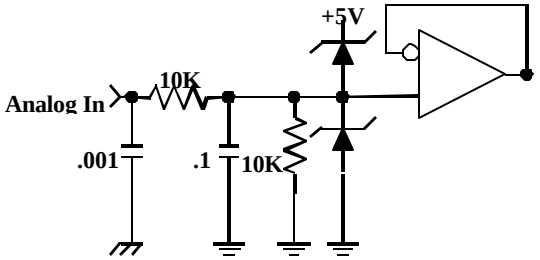
Outputs 1-4 (Sink)



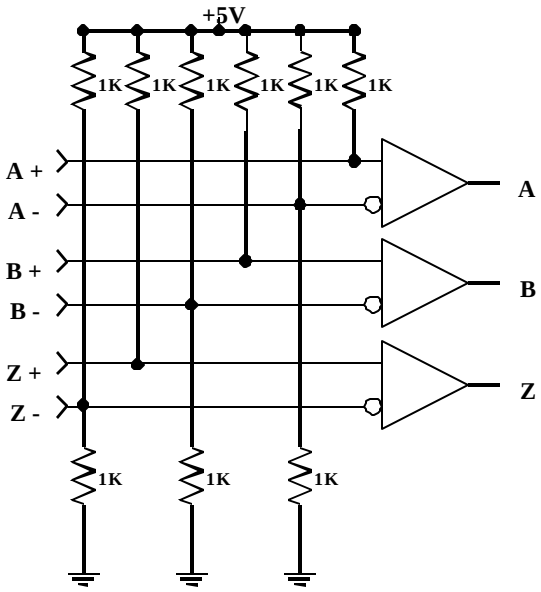
Outputs 5-8 (BCD Outputs)



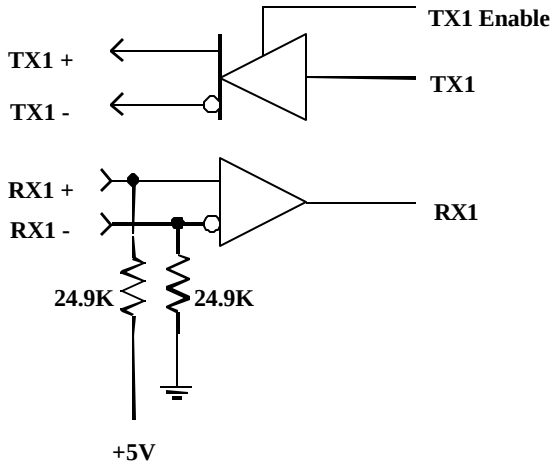
Analog Input



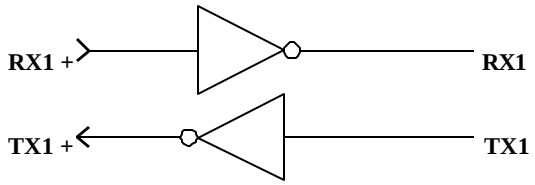
Encoder



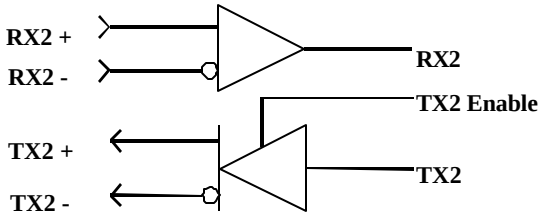
Port 1 TX/RX (RS485 position)



Port 1 TX/RX (RS232 position)



Port 2 TX/RX (RS485)



(This page left intentionally blank)

Section 5

Motion Controller Programming Interface

5.1 - PROGRAMMING

5.1.1 - General Description Of Programming

Programming of any sort requires planning and forethought. Programming your Controller is no exception. This section provides aids to facilitate your planning process.

5.1.1.1 - What is Programming?

A program is a list of discrete lines or command strings that, taken together in sequence, provide the information needed to get a machine to perform your predetermined sequence of instructions. These instructions can, in the case of Programmable Motion Controllers, cause the motor to move at certain speeds for given distances, read various inputs or set outputs, all used to accomplish different machine-related tasks.

5.1.1.2 - What's in a Program?

A program consists of many individual lines organized in a prescribed sequence. The **SS2000PCi** system uses an English language, BASIC-type computer programming language (SEBASIC). This makes it easy and intuitive to write and read machine control programs. SEBASIC supports many higher level language features, such as statement labels, subroutines, for-next and do-while loops for program flow control making it easy to write concise, well organized, easily debugged programs. Also, there are built in mathematics, Boolean functions and two dimensional array capability. Finally, the motion, I/O, and timing commands are easy to understand, remember and apply.

In addition to lines of program code , the controller uses and saves a series of set-up parameters. These parameters are set by the user in the Configuration & Setup section of the project.

5.1.1.3 - How is the Controller Programmed?

The programming environment called **Motion Controller Programming Interface** is supplied on a diskette. This software provides an easy to use environment for developing a user project. Detailed instructions on how to install this software on your PC are provided in this manual.

5.1.2 -What are Host Commands?

Host commands go straight from your input device (PC or terminal) to the controller. They allow parameters to be set or interrogated, motion to be started or stopped, and program execution to be started or stopped, etc..

5.1.3 - Memory Types and Usage

The controller uses two kinds of memory, volatile and non-volatile. RAM (Random Access Memory) is called **Volatile Memory** because when power is removed from the controller, all of the information in that memory is lost. The Controller stores the program variables in RAM.

The second kind of memory is **Non-Volatile Memory**, or FLASH memory. The information stored in this type of memory is not lost when power is removed. FLASH memory is used by the controller for storing the Operating System as well as the User Program.

A Controller program can have hundreds of lines of code. Code is simply an organized listing of program commands. Because of the wide variety of program commands it is impossible to state how many lines can be stored in the controller. The amount of free memory remaining can be obtained with the FREEMEM host command.

5.1.4 - How to organize your Project

A project consists of a Configuration & Setup section and the user program. The Configuration & Setup section allows access to project related parameters and conditions via folders. The user program performs your predetermined sequence of instructions.

A good program will consist of initialization, main program, Interrupt routines, Subroutines and Error Handler sections. The Interrupt routines, Subroutines and Error Handler sections are optional. A typical Program Development Block Diagram is provided in Figure 5.1.

5.1.4.1 - Initialization Section

Variable names and data types (Integer, Integer Array, Real and Real Array) are defined in this section. Also the condition's which will trigger the individual Interrupts (INTR1-INTR4) may be defined.

The values for ACCEL, DECEL, SPEED, FOLERR and WNDGS should also be set in this section. Comments may be added to make the program easier to follow and understand. An apostrophe ('), must be used at the beginning of the comment so that it will not be confused with the program statements.

Example Initialization Section:

```
STRING  Char$
INTEGER     a,b(100),c(10,3)
REAL    d,e(50),f(5,4)
‘           a       Integer variable
‘           b       Integer array - single dimension
‘           c       Integer array - two dimension
‘           d       Real variable
‘           e       Real array - single dimension
‘           f       Real array - two dimension
ON in(1)=1 INTR1
‘                   End of Initialization example
```

Note: all arrays are zero based. That is, the first element of the array has an index of zero, b(0) for example. The chart below shows two arrays: Array b is a single dimensional array of 101 elements. Array c is a two dimensional array of 44 elements in an 11 x 4 arrangement.

range	element size
b(0) - b(100)	101
c(0,0) - c(10,3)	44

The ON in(1)..... line tells the controller to Goto label INTR1 when in(1) is active. This condition is only checked after an INTRON1 command activates the interrupt checking.

This is only a simple example of an Initialization Section. The Programming Reference should be studied and understood before you write your own application.

5.1.4.2 - Main Program Section

The main program section should be placed just below the initialization section of the program. This section can use labels and any programming commands which may not be listed in the initialization section. Labels, however, cannot have the same name as programming commands. ***This section must be terminated with an END command***.

5.1.4.3 - Interrupt Routines

The Interrupt routine section is optional. Interrupt commands are powerful tools which instruct the program to check specified conditions after executing every program line. If the conditions are true, the program automatically jumps to a special interrupt routine which performs a desired program function. This section is only required when the ON .... INTRn command and INTRONn commands are used. These routines start with a specific interrupt label (INTR1, INTR2, INTR3 or INTR4) and ends with a RETURN command.

Interrupt conditions are only checked when the given interrupt is enabled. Each of the four possible interrupts are enabled using the associated INTRONn command. If the interrupt condition is true while the interrupt is enabled, then the routine INTRn will be executed. The INTROFFn command will disable checking of the interrupt condition.

Note: **n** is a value 1-4.

5.1.4.4 - Subroutines

The Subroutine section is optional. This section is only required if subroutine calls (GOSUB commands) are used by the project. Subroutines start with a label which is the subroutine name and ends with a RETURN command. The program statements in between can contain any valid programming command.

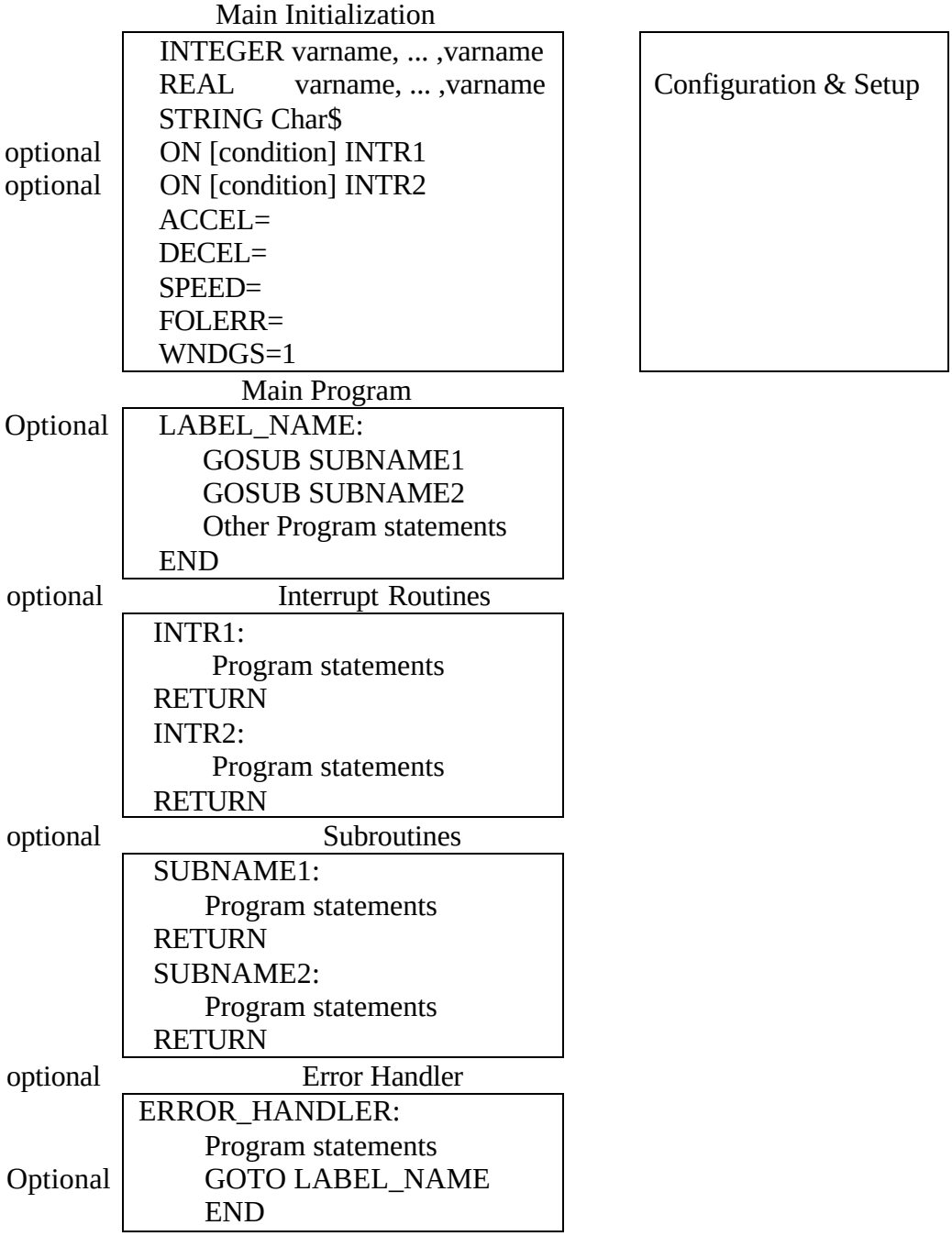
5.1.4.5 Error Handler

Error conditions encountered during program execution are handled in one of two ways:

1) The program jumps to a special routine labeled ERROR_HANDLER which must be written by the user specifically for the application. The **ERROR_HANDLER** label must be located at the start of the routine and the routine must terminate with an **END** or a **GOTO** statement. Any valid programming command with the exception of the **ON..INTRn** commands may be placed within the **ERROR_HANDLER** routine.

2) If no user **ERROR_HANDLER** routine exists, the program will terminate when an error condition occurs.

Figure 5.1
Typical Program Development Block Diagram



5.2 – MOTION CONTROLLER PROGRAMMING INTERFACE (MCPI)

5.2.1 - Software Installation

The Motion Controller Programming Interface (MCPI) provides the means by which an application can be fully developed and the controller can be operated using a personal computer (PC). The application can be written, compiled and downloaded to the controller, using the MCPI. In addition, a **Terminal Mode** is provided for operating the controller from your computer.

Minimum Computer Requirements:

1) Microsoft Windows® version 3.1 or later
2) Personal Computer with 80386, 80486, Pentium or higher microprocessor
3) 8 Megabytes of Random Access Memory (RAM)
4) 8 Megabytes of free hard disk space
5) VGA monitor and graphics card
6) Mouse or other suitable pointing device

Installation Instructions (Windows 3.1x):

1) If Windows is not already running type **WIN** at the Dos prompt, and press **ENTER.**
2) Insert the MCPI Program Disk into drive A: (or B:).
3) Click on the **File** menu in the Program Manager.
4) Select **RUN**... to display the Run Dialog box.
5) Type **A:setup** (or B:setup) and click **OK**.
6) The installation program will display the File Manager Setup screen. Follow the prompts on the screen to complete the installation.
7) After the program files have been installed, the installation will create a new Window group.
8) Remove the installation disk. This concludes the software installation.

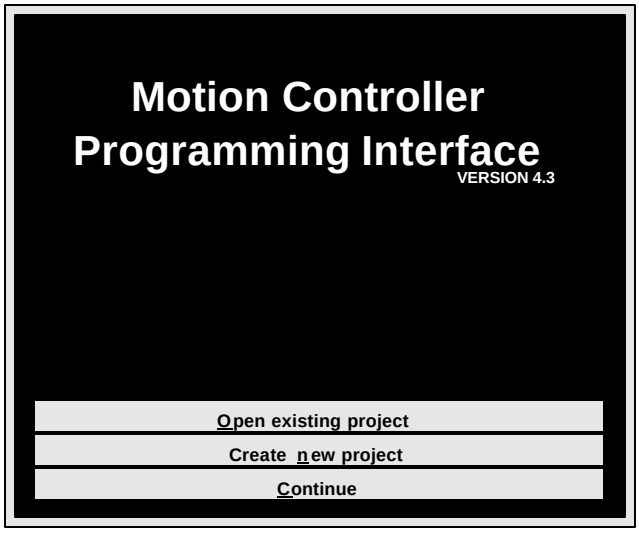
Installation Instructions (Windows 95):

1) If Windows is not already running type **WIN** at the Dos prompt, and press **ENTER.**
2) Insert the MCPI Program Disk into drive A: (or B:).
3) Click on the **Start** button.
4) Select **RUN**... to display the Run Dialog box.
5) Type **A:setup** (or B:setup) and click **OK**.
6) Follow steps 6 – 8 of Windows 3.1x installation.

5.2.2 - Starting the programming environment

1) If Windows is not already running, type WIN at the DOS prompt, and press **ENTER**.
2) Double click on the MCPI Icon.
3) The opening screen will appear.

5.2.2.1 - The MCPI program opening screen



Open existing project opens up an existing project.

Create new project creates a new project.

Continue enters the MCPI with no selection.

5.2.3 - Setting communication parameters

The MCPI PC program uses the computer's serial port to communicate with the controller. The PC program supports the use of four serial ports, (Com1-4). Three communication wires are required between the PC and the controller. These wires should be connected to the transmit (TX), receive (RX) and common (GND) as follows:

Computer	Controller
TX	RX+
RX	TX+
GND	GND

Notes: Consult your computer manual for the correct pin out assignment of its serial port. The factory default baud rate is 9600 baud. The controller supports 9600, 19200, and 38400 baud rates.

To use 19200 or 38400 baud rate with the controller do the following:

1) Set the 9600/User Baud rate switch on the controller to the User Baud position.

2) Load the user project, with the desired new baud rate programmed in the configuration & setup, into the controller .

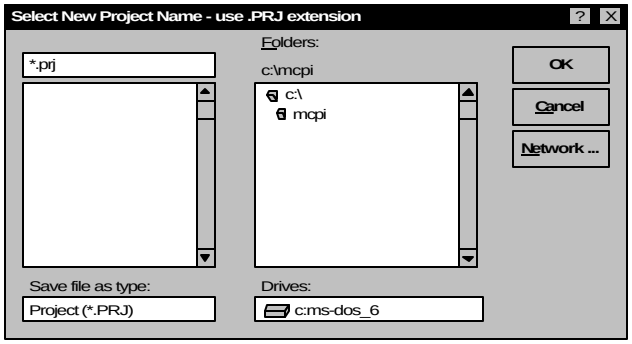
3) Set your terminal to the new baud rate using the System menu and selecting Terminal settings and then Com Port.

4) **Cycle power** on the unit to establish communications at the new baud rate. The baud rate switch is only read at power up or reset.

The serial communications to the controller can be tested by clicking on the **Terminal** command button and then on the **Software Revision** command button. The controller will send back the revision information if the setup is correct.

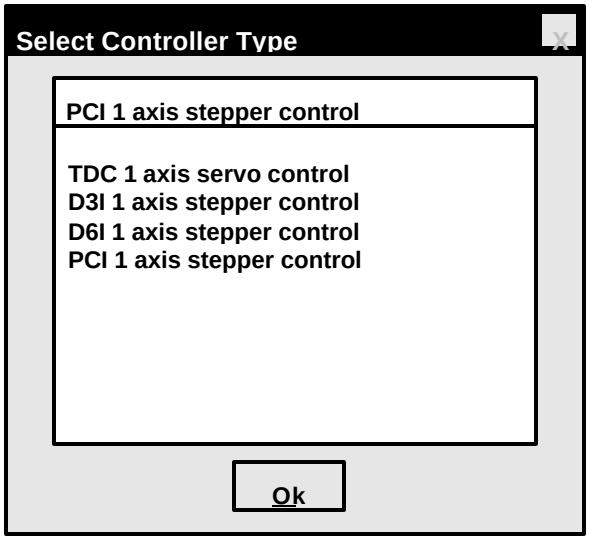
5.2.4 - Creating a new project

To create a new project either click on the **CREATE new project** command button on the Opening screen or the **New** item on the **Project** pull down menu.

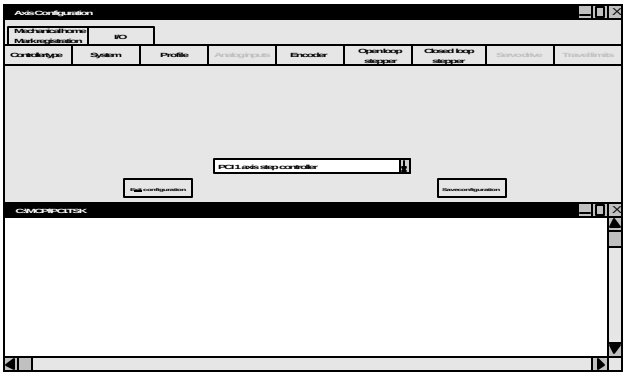


Enter the name of the project **with a .prj extension**. The directory for the project can also be selected at this time. To accept the name and directory click on the **OK** command button.

The controller type can now be selected by clicking on the desired selection and then clicking on the **OK** command button.



The controller type folder screen is now accessed. This screen allows access to the project folders by clicking on the desired folder tab.

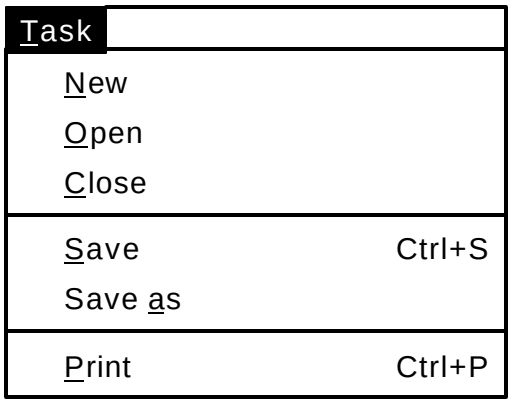


Save each folder that is changed by clicking on the **Save changes** command button. After completing all the changes to the configuration click on the **Exit configuration** command button.

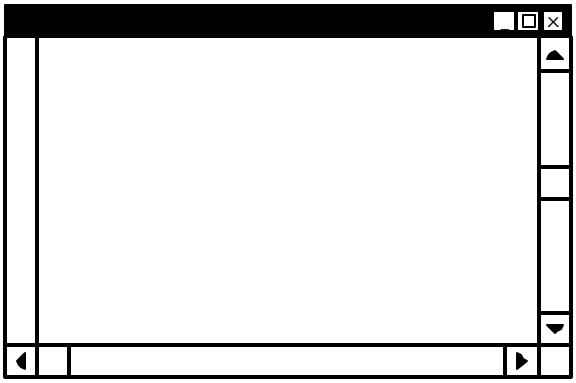
5.2.5 - The Task Editor

The Project program is created and edited using the Task Editor. To select the project to be edited click on the **Task** menu then either the **New** or **Open** item. The **New** selection allows a new task to be developed. The **Open** selection allows a previously developed task to be edited.

Task Menu Screen

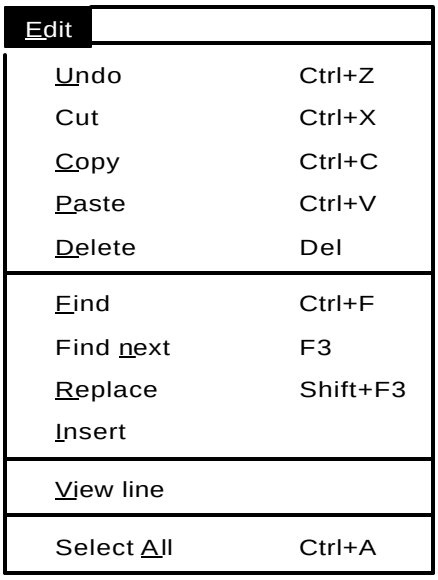


Task Editor Screen



The Edit functions can be accessed by clicking on the **Edit** menu and then clicking on the desired item. The Items and Actions for the **Edit** menu are described below.

Edit Menu



Undo (Ctrl+Z) undoes the latest action.

Cut (Ctrl+X) cuts (removes) the selected text and places it on the clip board.

Copy (Ctrl+C) copies the selected text and places it on the clip board.

Paste (Ctrl+V) pastes the contents of the clip board into the file where indicated.

Delete (Del) deletes the selected text.

Find (Ctrl+F) finds the occurrence of the selected text in the file.

Find next (F3) finds the next occurrence of the selected text in the file.

Replace (shift+F3) replaces one set of text with another set of text .

Insert Inserts a selected file at the current position.

View line go to selected line number.

Select all (Ctrl+A) selects all text.

Keyword checking enables or disables Keyword checking. If Enabled it Capitalizes keywords such as program commands and uses the selected colors for keywords and comments.

The Document setup functions are accessed by clicking on the **System** menu and then on the **Document** item. The Items and Actions for the **System** menu are listed below.

Fonts and colors selects the Font name, Font Style, Font size, background color, foreground color, Keyword color and Comment color. Some of these functions are duplicated on the Editor Tool Box.

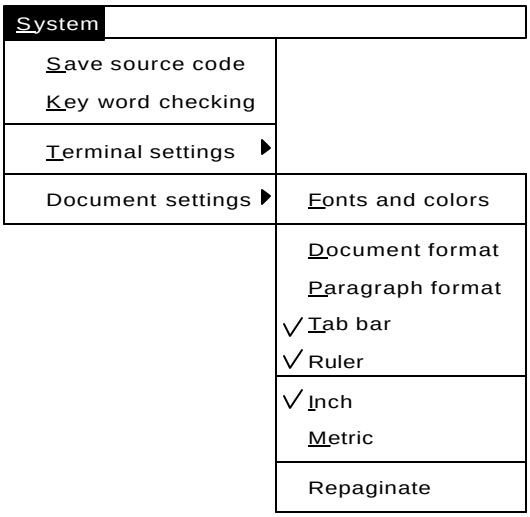
Document format selects the document width, height, margins and tab spacing. Some of these settings are duplicated on the Editor Tool Box.

Paragraph format selects the document margins, alignment, line spacing, tabulator type and tab spacing. These settings are duplicated on the Editor Tool Box.

Tab Bar displays the Tab bar when checked.

Ruler displays the ruler when checked.

System Menu

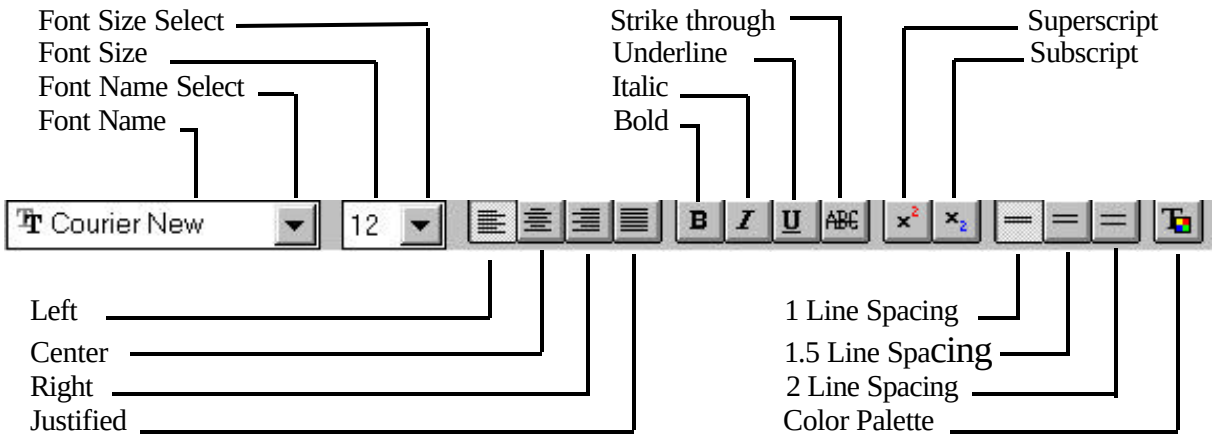


Inch selects the inch ruler when checked.

Metric selects the metric ruler when checked.

Repaginate repaginates the current task.

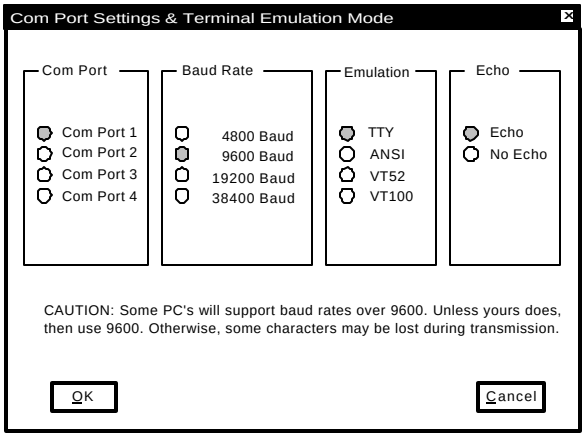
The Editor Tool Box, depicted below, can be used to modify the text on the Editor Screen. Fonts, type specifications, line spacing and text color can be modified using the Editor Tool Box.



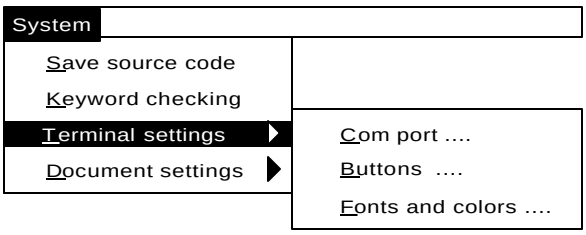
5.2.6 - Terminal Emulation

Terminal Emulation allows your PC to operate similar to a simple ASCII terminal. The user can enter/type Host commands (see Section 6.2), to perform operations or query the controller. These commands/queries will be immediately serviced by the controller, if possible.

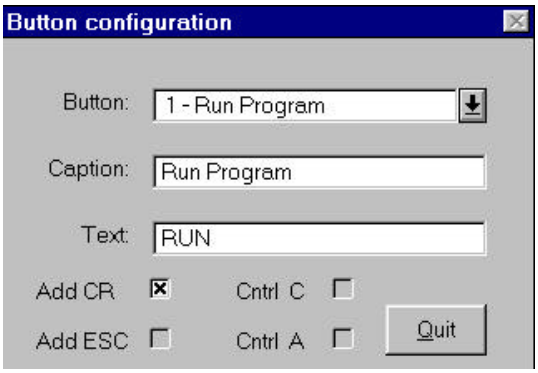
Before entering the Terminal Emulation environment, set up the communication port parameters by clicking on the **System** menu and then the **Terminal settings** item. Choose the appropriate Com port, baud rate, terminal emulation, and echo mode for the Com Port Screen by clicking ON one of the items (circles) in each section.



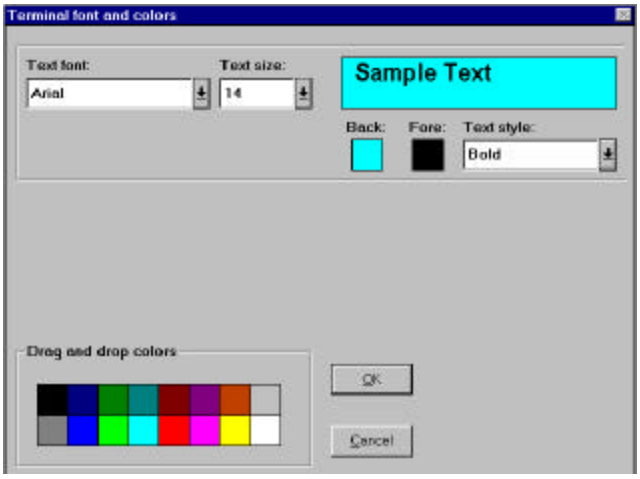
To program the buttons on the Terminal Emulation screen, Click on the **System** Menu and then on the **Terminal settings** item.



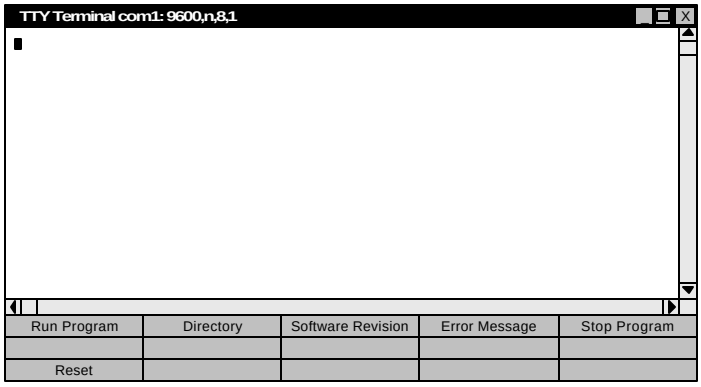
Click on the **Buttons** item which opens the Button configuration screen. Click on drop list arrow next to **Button** and select the button number to be programmed. Click on the **Caption** text box and enter the button caption text. Click on **Text** box and enter the command line text to be executed. If motion and program execution is to be stopped after the button's command is executed, click on **the Ctrl C** or **Ctrl A** check box. See the Host Command section of this manual for a more detailed description of **Ctrl A** and **Ctrl C**. If commands are to be allowed during program execution click on the **Add ESC** check box. Click on the **Add CR** check box if the **Ctrl C** or **Ctrl A** command are not selected.



To select the Font and Colors for the Terminal Emulation screen Click on the **System** Menu and then on the **Terminal settings** item. Click on the **Fonts and colors** item. Select the desired Font, Style, Font size, Background color and Foreground color for the Terminal Emulation environment. When finished, click on the O.K. button.



To enter the Terminal Emulation environment click on the **Terminal** command button.



5.2.7 - Configuration & Setup Folders

The folders for the configuration & setup screens are accessed by clicking on the **Configuration** command button. These folders allow project setup conditions to be programmed. A folder can be accessed by clicking on the folder tab.

Note: clicking on the Save changes command button saves the current folder data.

Clicking on the Exit Configuration command button can be used on any folder to exit the Configuration setup. If any of the items in the folder have been changed, a query will occur which will give the user the option of saving the folder data.

Clicking on another folder tab, will allow changes to the newly selected folder. The changes which have already been made will not be affected. This allows you to click between folders, set up the necessary parameters, and save only once before exiting the configuration screen. A description of each folder follows.

5.2.7.1 - System Folder

This folder defines the Drive Type, motor direction for a + motion and defines the units per motor revolution.

Drive type defines the type of drive operation. The choices are: **open loop stepper** or **closed loop stepper**. Open loop steppers do not have encoders, while closed loop steppers use encoders for position verification and/or correction.

The **Motor Direction** sets the motor direction for a + move. The choices are : **+ =cw motor direction** or **+= ccw motor direction**. The motor direction is as viewed from the rear of the motor.

The desired **Units per motor revolution** value should be entered. A unit is the method of measurement to be used, i.e. inches, mm, degrees, etc. This sets the number of user units for one motor revolution. Move distances and position values are in units, Speeds are in units/second and Accel/Decel values are in units/second^2.

System

	Drive Type		Motor Direction	Units per motor revolution
Axis 1	open loop stepper	↓	+ = cw motor direction	1.0

Examples:

1) Lead Screw

If a motor is directly coupled to a lead screw which has a 0.8" pitch, the units per motor revolution should be set to 0.8. The user may now write his program with distances in inches.

2) Rotary Table

The motor is connected through a 20:1 gearbox to a rotary table. If the user wishes to program motion in degrees of rotation on the rotary table, then the user units should be set to 360(motor degrees per motor rev) / 20(motor degrees per table degree) = 18(table degrees per motor rev). Now a move of 90 would result in 90 degrees of rotation of the table.

3) Conveyor

The motor is connected through a 10:1 timing belt pulley reduction to a drive roller that drives a conveyor. Every turn of the drive roller advances the conveyor by 0.5 feet. If the user wishes to program motion in feet of movement of the conveyor, he must set the user units to 0.5(conveyor feet per drive roller rev) / 10(motor revs per roller rev) = 0.05 (conveyor feet per motor rev).

5.2.7.2 - Profile Folder

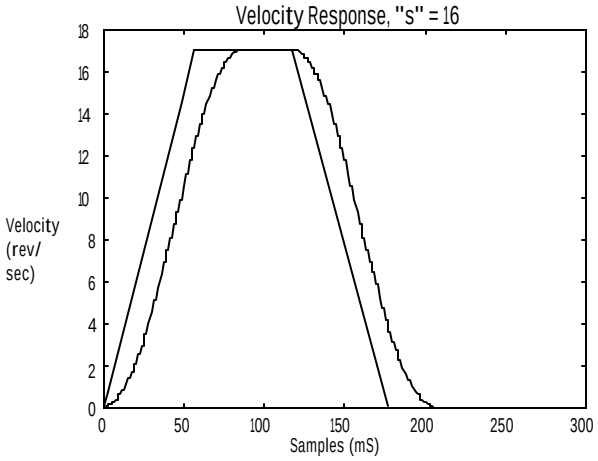
This folder selects the motion profile, maximum acceleration rate, maximum speed and Delay after motion.

Profile

	Motion profile		Max. acceleration (units/sec^2)	Max. speed (units/sec)	Delay after motion (sec.)
Axis 1	trapezoidal	↓	200.0	50.0	0.05

Motion Profile determines how the motor's speed changes. Speed changes require a period of accel/decel to increase/decrease the motor's speed. The "Motion Profile" determines how the accel is applied. There are 32 choices, and a profile setting of 1 results in a "Trapezoidal" profile. This profile yields the minimum move time. Settings 2 - 32 yield "S-curve" profiles with varying degrees of smoothing. The higher the profile setting, the more "S" like the profile. Move times with profile settings 2 - 32 are from 2 to 62 ms longer respectively than those with a setting of 1. The "S-curve" profiles usually results in smoother motion at the expense of longer move times.

Move times can be shortened, however, by raising the accel, decel, and/or speed of the move.



Max. acceleration sets the maximum allowed acceleration or deceleration rate in units/second^2. This value is also used to decelerate motion to a stop when a fault such as a travel limit occurs.

Max. Speed sets the maximum allowed target speed in units/second. Speed, Accel and Decel values can be reset within a program as long as the value used is less than or equal to the max speed and max accel respectively.

Delay after motion sets the minimum time, in seconds, between two moves.

5.2.7.3 - Encoder Folder

Used to set the Encoder direction and Encoder resolution.

Encoder direction **determines how the encoder rotation direction is interpreted. The choices are:** normal direction **or** reverse direction.

Encoder line count defines the encoder resolution in lines. An Encoder with 1000 lines will provide 4000 counts/revolution, or quadrature counts. Set this value to the encoder line count of the motor.

Encoder

	Encoder direction	Encoder line count (lines / rev)
Axis 1	normal direction	500

5.2.7.4 – Open Loop Stepper Folder

Sets the Steps per motor revolution, Motor standstill current, Motor boost current and motor current delay for an open loop stepper drive.

Steps per motor revolution defines the resolution on the stepper drive connected to the controller.

Motor standstill current sets a percentage of motor current when the motor is at standstill. The choices are **normal (100%)**, **reduced (50%)** and **off (0%)**.

Motor boost current sets a percentage of motor current when the motor is running. The choices are **normal (100%)** and **boost (150%).**

Motor current delay specifies the time delay between current modes in seconds. This allows for the drive to respond to the change in current level as a result of the BOOST or REDUCE commands (see Program Command section).

Open Loop Stepper

	Steps per motor revolution	motor standstill current	motor boost current	motor current delay (sec)
Axis 1	2000.0	normal (100%)	normal (100%)	0.05

5.2.7.5 – Closed Loop Stepper Folder

Sets the Steps per motor revolution, Motor standstill current, Motor boost current, motor current delay, for a closed loop stepper drive.

Steps per motor revolution see Open Loop Stepper Folder for description.

Motor standstill current see Open Loop Stepper Folder for description.

Motor boost current see Open Loop Stepper Folder for description.

Motor current delay see Open Loop Stepper Folder for description.

Error action selects what action, if any, is taken by the controller when the commanded motor position does not match the encoder position within the range set by the **FOLERR** command (see programming commands). This is also referred to as a stall condition. Once the **FOLERR** range is exceeded one of four things can happen according to the **Error Action** selected.

If **Error action** is **disabled**, the controller takes no action.

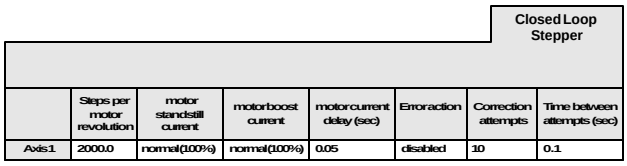
If **Error action** is **stop on error**, the motor will stop and a controller error will result (see ERR command). The fault light will illuminate.

If **Error action** is **correct on error**, separate correction attempts (moves) will be commanded to try to re-align the motor. The user may specify how many correction attempts will occur, and the time between attempts. If after the specified maximum number of correction attempts the motor still is not aligned, motion stops and a controller error will result.

If **Error action** is **restart on error**, the entire move is restarted. The motor returns to the starting position of the move in progress, and attempts to repeat the move. If during this repeat cycle the motor stalls, the motor will return to the start position and retry the move. Each stall and restart counts as a correction attempt. This continues until the motor reaches the desired position, or the maximum number of **correction attempts** is reached. In the case of the latter a controller error results and the fault light illuminates.

Correction attempts specifies the maximum number of consecutive attempts allowed when **error action** is set to **correct on error** or **restart on error** mode and the motor stalls.

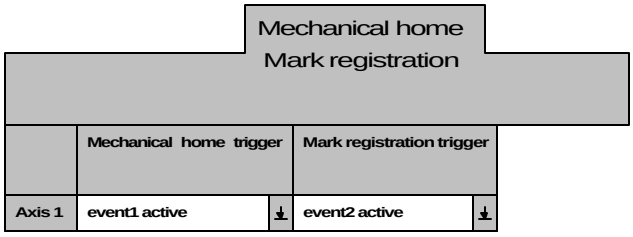
Time between attempts specifies the time between correction attempts when **error action** is set to **correct on error** or **restart on error** mode and the motor stalls.



5.2.7.6 - Mechanical Home & Mark Registration Folder

This folder specifies the trigger for the mechanical home (MOVEHOME) and the mark registration cycle (MOVEREG).

Mechanical Home trigger & Mark Registration trigger specifies the trigger for the cycle. There are two inputs EVENT 1 and EVENT 2 which can be used as a trigger. The trigger combination for Mechanical home and Mark registration are: **event 1 active**, **event 1 inactive**, **event 1 active & encoder marker**, **event 1 inactive & encoder marker**, **encoder marker active**, **encoder marker inactive**, **event 2 active** and **event 2 inactive**.



5.2.7.7 - I/O Folder

This folder is used to assign inputs 3-7 as general purpose inputs or as dedicated functions. In addition, the auto program start can be enabled or disabled and the host baud rate for the User position can be programmed.

Input 3 (+limit) and **Input 4 (-limit)** can be configured as hard limit inputs or as general purpose inputs. As limits, the active signal level can also be configured as either active on switch closing or active on switch opening.

Inputs 5, 6 & 7 can be *collectively* configured as general purpose inputs or as **Run**, **Clear** and **Feedhold** functions respectively.

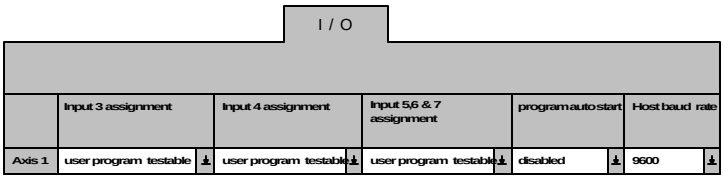
Run - Program execution is started using the **Run** input or the host "**RUN**" command. At power up an active **Run** input will start program execution unless the **Clear** input is inactive. Following the initial power up, an inactive to active transition on the **Run** input does the following: If the **Feedhold** state is set, and the **Feedhold** input is inactive then **Run** clears the feedhold state, otherwise **Run** simply starts the user program if the **Clear** input is active.

Clear - An **inactive** clear input will stop program execution. ***The Clear input has the opposite polarity from other inputs, and the Clear function is enabled if the pin is left floating.*** If motion is occurring, the motor is decelerated to a stop using the maximum accel/decel value, and the Feedhold state is cleared. Run and Feedhold commands are ignored as long as the clear input is inactive.

Feedhold - An active Feedhold input sets the Feedhold state. When the Feedhold state is set, motion is decelerated to a stop using the programmed DECEL. When the Feedhold state is cleared by a Run command the motion is resumed. The Feedhold state remains cleared if the Clear input is inactive.

The **Program auto start** feature may be enabled for a power-on condition or reset command. The user may enable this feature to allow the program to start executing automatically upon a power up or reset condition.

The **Host baud Rate** may be set to 9600, 19200 or 38400. Once the corresponding Project is downloaded into the controller, this rate will be used for serial communications to the Host (usually a PC) if and only if the Baud Rate switch on the controller is set to the User Baud position. After downloading, the new baud rate will take effect upon power up or reset. If the baud rate switch is in the 9600 position, all communications will occur at 9600 Baud.



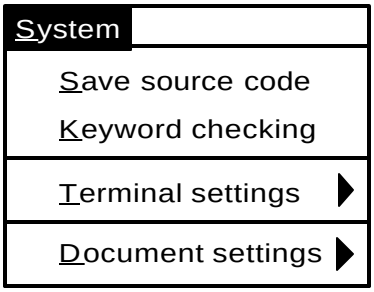
5.2.8 - Preparing User Project for Execution

In order to execute a project program it must first be compiled and then Downloaded to the controller. The project source code can be recovered from the controller as well.

5.2.8.1 - Project Source code

The Project Source Code is the English version of the user=s program. If the user=s program needs to be uploaded from the controller at any time, ASave Source Code≅ must be enabled. The Source code of a project can be saved in the controller. However, the source code uses up program memory in the controller. The selection for source code saving is accessed by clicking on the **System** menu. The Save source code setting can be toggled by clicking on the **Save source code** item.

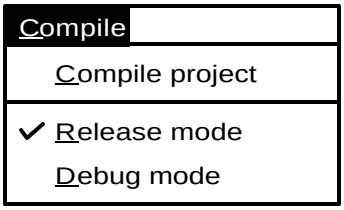
Note: Saving source code in the controller requires a lot of program memory. If the user=s program is extremely long it may not be possible to save the source code. See the FREEMEM command for more information.



5.2.8.2 - Setting Project Debugging

The users program may be compiled in Debug mode. Debug mode allows the user to step through their program and monitor its operation and the status of I/O, variables, motion, etc. Once the program has been compiled the Debug environment can be entered (See Section 5.2.12 Debug Environment).

To set the debugged mode click on the **Compile** menu and then on the **Debug mode** item. The project must now be compiled and downloaded before task debugging can begin. To cancel the debugging mode selection click on the **Compile** menu and then the **Release mode** item. To complete this cancellation the project must now be compiled and downloaded.

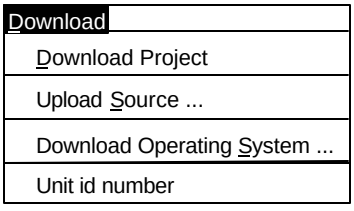


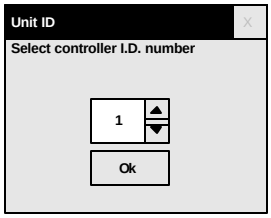
5.2.8.3 - Compiling a Project

Whether the project is new, or changes have been made to the task or configuration, it MUST be compiled BEFORE DOWNLOADING for it to be stored and implemented in the controller. Compiling converts the users task and configuration to machine code that the controller can understand. A project can be compiled by clicking on the **Compile** Command button or on the **Compile** menu and then the **Compile project** item.

5.2.8.4 – Selecting the Controller Unit ID

The controller unit ID can be selected by clicking on the **Download** menu and then the **Unit id number** item. The unit ID can now be selected using the spin controller and then clicking on the **Ok** command button.



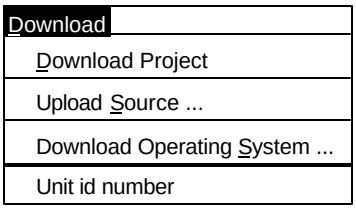


5.2.8.5 - Downloading a Project

A project can be downloaded to an active unit with or without its source code by clicking on the **Download** command button or clicking on the **Download** menu and then the **Download project** item. The project is sent to the active unit only. See Section 5.2.8.4.

5.2.8.6 - Uploading Source Code

A projects source code can be uploaded from the controller to the PC by selecting the **Upload Source** item in the **Download menu**. A project can be uploaded from the controller ONLY if it had previously been saved in the controller. See section 5.2.8.1. The Upload is from the active controller when there is more than one unit on a daisychain. See Section 5.2.8.4 to select the controller ID.

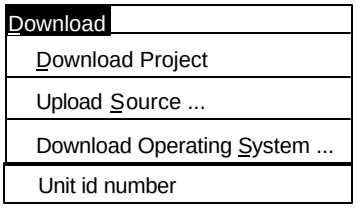


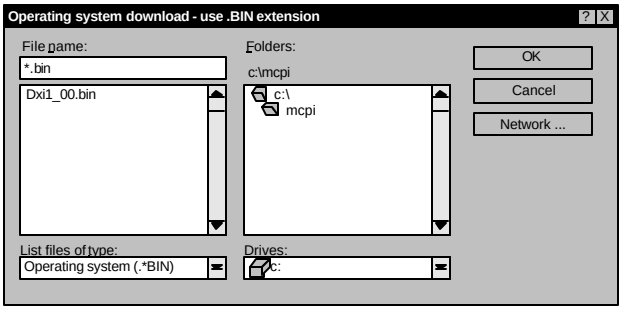
5.2.9 - Downloading an Operating System

Although the unit comes with an operating system installed, new operating system software can be downloaded by clicking on the **Download menu** and then the **Download Operating System** item.

The operating system file, with an extension **.bin**, can now be selected by clicking on the desired file name. To start the operating system download procedure click on the **OK** command button.

Note: The file names for the different controllers start with the following letters: **Pci** for SS2000Pci, **tdc** for the TDC controller and **DxI** for the SS2000DD3i/6i controller. If on a daisychain, the appropriate unit ID number must first be selected.





5.2.10 - Other Menus

The MCPI menus are pull down menus. Clicking on a menu shows an itemized list of operations allowed for that menu. The menus are: Project, Task, Edit, Compile, Download, Utility, System, Window and Help. To select a menu operation, click on it or press the Alt key and the underlined character in the title simultaneously.

5.2.10.1 - Project Menu

This menu allows you to create a new project, open an existing project, save a current project, add or remove a task from a project, open the configuration & setup environment, print a current project, or exit the MCPI programming environment.

New is used to create a new project.

Open is used to open up an existing Project.

Save is used to save the current project.

Save as is used to save the current project under a new name.

Remove task is used to remove a task file from an open project.

Add task is used to add a file to a current project.

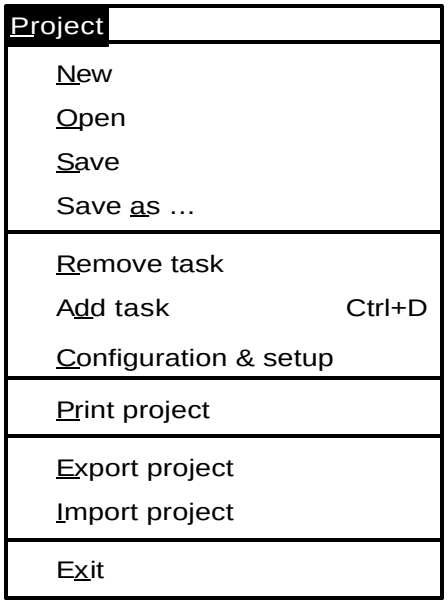
Configuration & setup is used to edit the Configuration & setup folders.

Print project is used to print a current project's information.

Export project is used to export a current project to another drive or directory.

Import project is used to import a selected project from another drive or directory into the MCPI Environment.

Exit is used to exit the MCPI programming Environment.

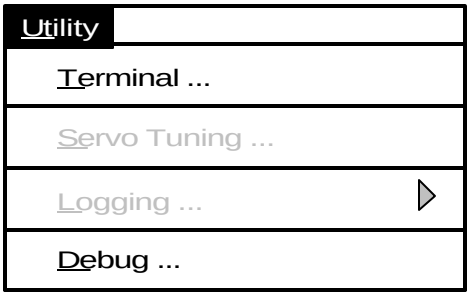


5.2.10.2 - Utility Menu

This menu allows reselection of terminal mode emulation, data logging, servo tuning, or program debugging.

Terminal starts terminal emulation mode. This allows direct communication with the controller.

Debug starts program task debugging.



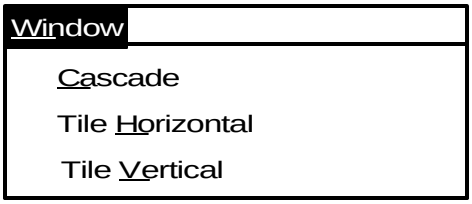
5.2.10.3 - Window Menu

This menu selects the windows format for the open windows.

Cascade cascades the open windows.

Tile Horizontal tiles the open windows Horizontally.

Tile Vertical tiles the open windows Vertically.



5.2.10.4 - Help Menu

This menu provides help on program commands, technical assistance and displays the MCPI software version.

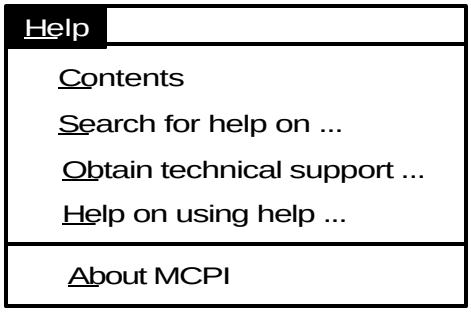
Contents list the help topics.

Search for help on lists the help items and descriptions.

Obtaining technical support provides application assistance telephone numbers.

Help on using help provides help on how to use Help.

About MCPI provides the MCPI version number.



5.2.11 - Project Command Buttons

The MCPI command buttons allow the selection of the Configuration & Setup folders, compilation of a current project, downloading of a current user project, selecting the Terminal Emulation environment, selecting the servo tuning environment, or selecting the program debugger environment.

Configuration enters the configuration & setup environment.

Compile project compiles a current project.

Download project downloads a current project.

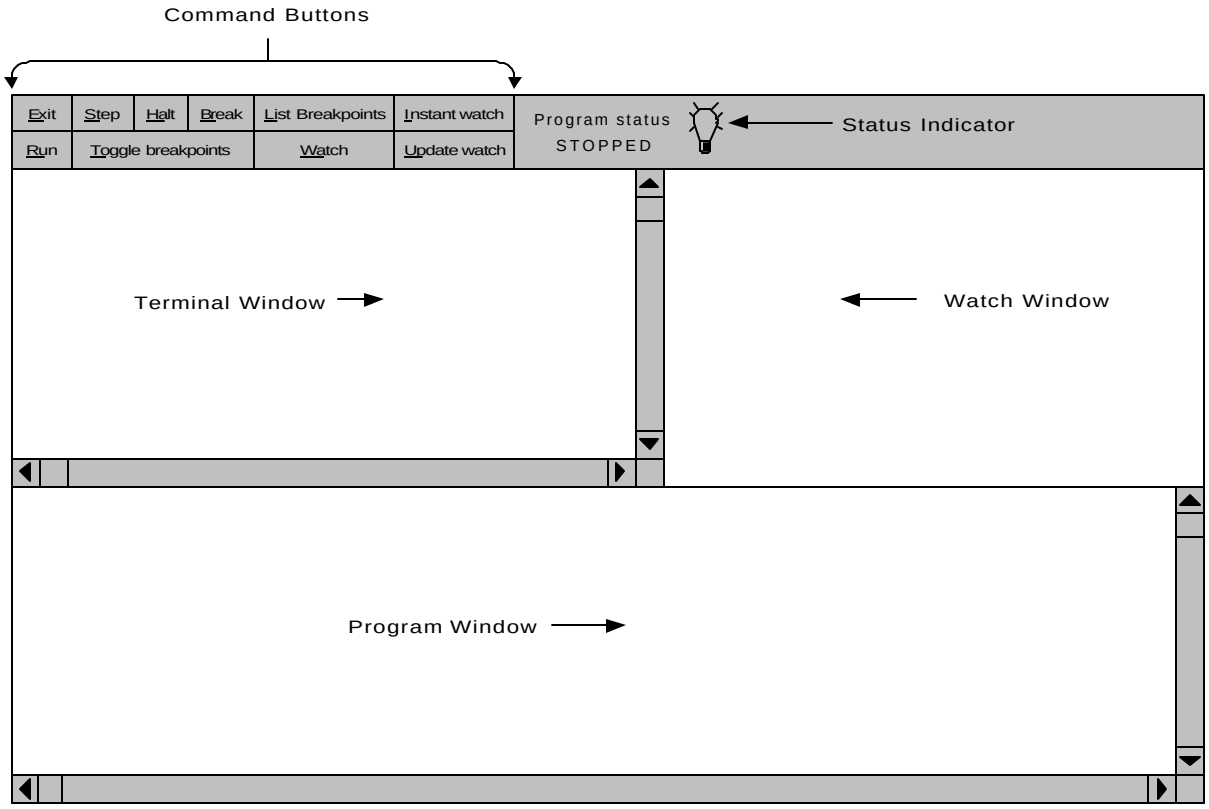
Terminal enters the Terminal emulation environment.

Debug enters the Program Debugger Environment.

Configuration	Compile project	Download project	Terminal	Servo tuning	Debug

5.2.12 - DEBUG Environment

A project that is loaded into the controller can be debugged if the project has been compiled in Debug mode and downloaded. (See previous section for Setting Project Debugging) The project to be debugged must be open. To enter the debugger environment click on the **Debug** command button. This environment consist of a Program status indicator, Command buttons for **Exit, Run, Halt, Toggle breakpoints, Watch, Update Watch, Step, Break, List Breakpoints,** and **Instant Watch** , a **Terminal** window, a **Watch** window and a **Program** window.



5.2.12.1 - Debug program execution

A program can be executed in several different ways from the Debug Environment. Single line execution of the current line can be initiated by clicking on the **Step** command button. The > symbol preceding the line number indicates the line to be executed. The program can be executed to the next breakpoint encountered or the end of the program by clicking on the **Run** command button. A Running program can be halted by clicking on the **Halt** command button. A program that is running can also be placed in the Single line execution mode by clicking on the **Step** or **Break** command button.

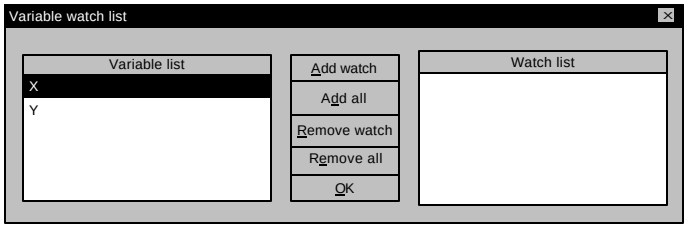
Note: The program status indicator shows the status of program execution. The only time this status will indicate Stopped is when the program is halted or has executed an end statement in the program. The indicator is green for running and red for stopped.

5.2.12.2 - Breakpoint Setting/Clearing

Up to five breakpoints can be set in debug mode. To change the breakpoint setting of a line, click on the desired line and click on the **Toggle breakpoints** command button. When a line is set as a breakpoint, a **(BRK)** indicator will precede the line. The breakpoint line numbers can be listed or cleared by clicking on the **List breakpoints** command button and then the appropriate command button.

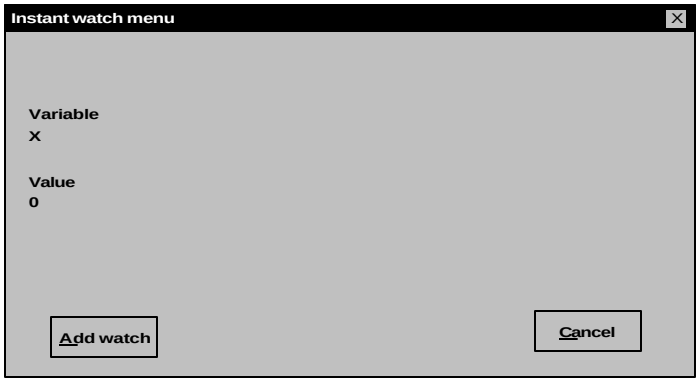
5.2.12.3 – Variable Watch

Variable watch allows the programmer to view the values of selected variables. To add or remove a watch variable from the watch window click on the **Watch** command button.



To add all the variables to the watch list click on the **Add all** command button. To add a specific variable to the watch list, select the variable in the Variable list and then click on the **Add watch** command button. To remove all variables from the watch list click on the **Remove all** command button. To remove a specific variable from the watch list, select the variable in the Watch list and then click on the **Remove watch** command button. To return to the Debug Environment screen click on the **Ok** command button. The variable in the watch list will appear in the Watch Window and its current value will be displayed.

Another method of watching a variable is to highlight the variable and then click on the **Instant Watch** command button. The variable name and value will be displayed. This variable can be added to the watch window by clicking on the **Add watch** command button.



5.2.12.4 - Terminal Window

The terminal window allows host command execution without leaving the Debug Environment. The Terminal Window is selected by clicking inside the Terminal window. A blinking cursor indicates that the Terminal window is selected for host commands.

5.2.12.5 - Exit Debug Environment

The debugger environment can be exited by clicking on the **Exit** command button.

(This page intentionally left blank)

Section 6.0

Software Reference Guide

6.1.1 - Programming Commands Grouped By Function

Motion

Trajectory Parameters

I/O

Over Travel Limit

		Page
HARDLIMOFF	Disables hard limits.	84
HARDLIMON	Enables hard limits.	85
REGLIMIT	Sets or returns the movereg limit distance.	111
SOFTLIMNEG	Sets or returns the absolute negative travel limit position.	112
SOFTLIMOFF	Disables soft limits.	113
SOFTLIMON	Enables soft limits.	114
SOFTLIMPOS	Sets or returns the absolute positive travel limit position.	115

Time Functions

TIMER	Sets or returns timer value.	118
WAIT	Wait for the period of time to expire.	120

Program Flow Control

DO...EXIT DO...LOOP... LOOP...UNTIL...WHILE	Begin a repeatable a block of statements.	72
END	End of program.	74
FOR...TO...EXIT FOR...NEXT	Begin a repeatable block of statements.	80
GOSUB...RETURN	Branch to a subroutine and return.	82
GOTO	Branch unconditionally to the specified label.	83
IF..THEN..ELSE..END IF	Begin a conditional block of statements.	87

Interrupt

INTROFFn	Disable interrupt n, where n is 1-4.	93
INTRONn	Enable interrupt n, where n is 1-4.	93
ON...INTRn	On condition go to interrupt n, where n is 1-4.	102

Miscellaneous

DEFINE	Defines a symbolic name to be a particular string of characters.	70
ERR	Return error code number.	75
INCLUDE	Includes a file name with define statements in a user task.	90

Boolean Expression Operators

AND	Logical conjunction operator.	66
NOT	Logical complement operator.	101
OR	Logical inclusive OR operator.	104

String Manipulation

		Page
ASC	Returns the ASCII code of character.	66
CHR$	Returns a one character string for the given ASCII code.	68
GETCHAR	Waits for a character to be received via the serial port.	81
HEX$	Returns the hex string of an integer.	86
HVAL	Returns the hex value of a string.	86
INCHAR	Returns a character from the serial port.	89
INPUT	Reads a line of data from the serial port.	91
INSTR	Returns the first occurrence of a character in a string.	92
LCASE$	Converts a string to lower case letters.	94
LEFT$	Returns the leftmost characters of a string.	95
LEN	Returns the number of characters in a string.	95
MID$	Returns the designated middle number of characters of a string.	96
PRINT	Transmit data via the serial port.	106
PRINT USING	Print string characters or formatted numbers.	107
RIGHT$	Returns the rightmost characters of a string.	111
STR$	Returns a string representation of a numeric expression.	117
STRING$	Returns a string of characters.	118
UCASE$	Converts a string to upper case letters.	119
VAL	Returns the value of a string.	119

Relational Operators

=	equal to	62
<	less than	62
<= or =<	less than or equal to	62
<>	not equal to	66
>	greater than	62
=> or >=	greater than or equal to	62

Arithmetic Operators

+	addition	62
-	subtraction or unary minus	62
*	multiplication	62
/	division 62	

Variable Definitions

INTEGER	var, ... , var	63
REAL	var, ... , var	63
INTEGER	var(x), ... , var(x,y)	63
REAL	var(x), ... , var(x,y)	63
STRING	$var,…, $var	63

Note: Arrays up to two dimensions are supported. Values for x and y must be greater than zero.

6.1.2 Programming Commands Summary (alphabetical list)

6.1.3 - SEBASIC Conventions

A BASIC-like language ("SEBASIC") which conforms to most of the rules and conventions of modern implementations of the BASIC programming language, such as "QuickBasic", etc. The following is a summary of the considerations to be used in writing your programs.

6.1.3.1 - Arithmetic Operators

The SEBASIC arithmetic operators, listed in order of precedence, are:

Operator	Function
-	Negation
*, /	Multiplication and division. See BASIC DATA TYPES section for notes on division.
+, -	Addition and subtraction

Parentheses change the order in which arithmetic operations are performed. Operations within
parentheses are performed first. Inside parentheses, the usual order of operation is maintained.

NOTE: Squaring and exponentiation are not supported; use multiplication to perform these operations.
Example: to calculate X^3, use X*X*X.

6.1.3.2 - Logical Operators

These operators are used in boolean expressions. The logical operators in SEBASIC, listed in order of precedence, are as follows:

Operator	Use
NOT	NOT<term> a false term, results in the boolean expression being true.
AND	<term> AND <term> both terms must be true, results in the boolean expression being true.
OR	<term> OR <term> either term being true results in the boolean expression being true.

Logical operators perform tests on multiple relations, bit manipulations, or Boolean operations,
and return a true (one) or false (zero) value to be used in making a decision.

6.1.3.3 - Relational Operators

Relational operators are used to compare two values. The result of the comparison is either
"true" (one) or "false" (zero). This result can then be used to make a decision regarding program flow.

Operator	Relation	Expression
=	Equality *	X=Y
<>	Inequality	X<>Y
<	Less than	X<Y
>	Greater than	X>Y
<=	Less than or equal to	X<=Y
>=	Greater than or equal to	X >=Y

* The equal sign (=) is also used to assign a value to a variable.

6.1.3.4 - BASIC Data Types

Three basic data types exist: **REAL, INTEGER,** and **STRING** values.

The following are examples of some REAL values:

+1.524 -100.1 2.1e-4

Note that "e" or "E" may be used as the exponential operator, i.e. power of 10. For example 5004.1 may also be represented as 50.041E2. In this case 50.041 is the mantissa and the exponent is 2. The mantissa of a real number is limited to a 15 digit representation. If the + is omitted, the value defaults to a positive number.

**The range for REAL numbers is +/-
1.7 E ± 308 (15 digits).**

The following are examples of some INTEGER values:

+1 -100 -3487

If the + is omitted, the value defaults to a positive number.

The range for INTEGER numbers is ± 2,147,483,647.

String values can be any ASCII character. A list of ASCII characters is provided in Section 9.0 Glossary.

Rules For Integer Division:

When the division operator, "/", is used to divide integers, some rules must be followed to achieve the expected results from the calculation. If fractional information is to be included in the result of the division of two numbers, at least one of them MUST be a REAL number. If both are INTEGER, the operation will produce an INTEGER result. Some programming examples are shown below.

```
INTEGER   num1,denom1      'declare INTEGER variables
REAL      answer1,answer2,
          denom2           'declare REAL variables

begin:                     ' begin program
num1 = 10                  ' set INTEGER num1 equal to 10
denom1 = 4                 ' set INTEGER denom1 equal to
4
denom2 = 4                 ' set REAL denom2 equal to 4
answer1 = num1/denom1      ' divide num1 by denom1
answer2= num1/denom2       ' divide num1 by denom2
end                        'end program
```

In this case the value of answer1 will be 2. This is because num1 and denom1 were declared as INTEGER numbers. The value of answer2 will be 2.5, as expected. This is because denom2 was declared as a REAL variable. When assigning a variable to a number which is represented in the code by a fraction, the numerator or denominator MUST use a decimal point if the result requires fractional information. For example:

```
REAL x      'declare x as a REAL variable

x = 10/4    'x will be equal to 2  since 10 and 4 without a
            'decimal point are integers

x = 10.0/4  'x will be equal to 2.5 because of the decimal
            'point in 10.0
```

Note: All variable names and program labels must begin with a letter A-Z.

6.1.3.5 - Case Sensitivity In Statements & Commands

Some programming statements and commands are case sensitive; others are not. The following table defines case sensitivity in SEBASIC:

BASIC LANGUAGE ELEMENT	CASE SENSITIVE?	MAX. LENGTH (characters)
Label	No	80
Variable name (symbolic constant)	No	80
BASIC keyword	No	N/A

The **Host** commands are not case sensitive; that is, upper and lower case letters can be used interchangeably.

6.1.3.6 - Calculations Using Trajectory Parameters And Variables

Caution must be used when performing calculations based on the Trajectory Parameters, **ABSPOS, ACCEL, DECEL, DIST, ENCPOS, ENCSPD**, and **SPEED**. Comparisons of values returned directly from reading these parameters or values of variables calculated from these parameters may not always yield the expected results. The reason for this is that digital systems have inherent resolution limitations. In the case of this system, the actual position of the motor shaft at any time can only be represented within one microstep (1/64 full step). If the user is programming in units, the actual position is calculated based on the number of microsteps per user unit.

If the user program must compare a calculated value to any of the trajectory parameters (such as ENCPOS), or to variables which have been derived from them, then the use of the equals operator is NOT RECOMMENDED. Using greater than, > , less than, <, greater than or equal to, >=, or less than or equal to, <= is therefore recommended for proper operation of the program. In fact, ANY calculated variables may have a very small fractional portion which may cause problems when comparing them to be exactly equal to either another variable or a number entered in the code.

6.1.3.7 Program Comments

An apostrophe (') in a program line prevents a line from executing and allows program comments/documentation. All text to the right of the ' to the end of line is not considered part of the command during execution.

EXAMPLES:

```
'MOVEI=10      >The program will not execute this line
MOVEI=100      >The program will execute this line
```

6.1.4 Programming Commands - Alphabetical Listing

ABSPOS

Trajectory Parameters

ACTION: Sets or returns the commanded absolute position of the motor.

PROGRAM SYNTAX:
ABSPOS=expression
ABSPOS - used in an expression

REMARKS:
ABSPOS=expression
Sets the absolute position in units.

ABSPOS - used in an expression
Evaluates and returns the current absolute position.

ABSPOS represents the commanded motor position, and can only be set while no motion is occurring. Setting ABSPOS during motion, causes the program to be trapped at the ABSPOS instruction until the motion completes. When ABSPOS is set or read, the internal representation is limited to ± 2,147,483,647 encoder counts. Setting ABSPOS also sets ENCPOS (encoder position) to the same value. ABSPOS and ENCPOS are initialized to 0 at power up. ABSPOS is also set at the end of a MOVEHOME command to the distance traveled from the home cycle trigger. Reading ABSPOS returns the actual commanded position in user units.

EXAMPLES:
ABSPOS=2
sets absolute position to 2 units.

a=ABSPOS
returns the ABSPOS position value to variable "a".

ACCEL

Trajectory Parameters

ACTION: Sets or returns the acceleration rate of the motor.

PROGRAM SYNTAX: ACCEL=expression
ACCEL - used in an expression

REMARKS: ACCEL=expression
Sets the acceleration rate in units/sec^2.

ACCEL - used in an expression
Evaluates and returns the present acceleration rate.

Specifies the rate at which the motor speed is increased. Specifying a 0 or negative value will result in error code 6. Specifying a value greater than **Max Accel,** set in the system *Configuration and Setup,* will result in ACCEL being set to the **Max Accel** value. At power up and each time a program is run, ACCEL is initialized to 50% of the **Max Accel** value. ACCEL can be set during motion, but the new setting will not be used until the next motion. Reading ACCEL returns the most recent setting. The lowest allowable acceleration rate is 0.07276 rev/sec^2. The highest allowable acceleration rate is 1192 rev/sec^2.

EXAMPLES: ACCEL=2
Sets acceleration rate to 2 units/sec^2.

a=ACCEL
returns the acceleration value to variable "a".

ANALOG

I/O Functions

ACTION: Returns the analog input value in volts.

PROGRAM SYNTAX: ANALOG - used in an expression

REMARKS: ANALOG
Evaluates and returns the present analog input voltage in volts. This value may vary for successive reads, but will stay within the accuracy listed in the Hardware specification section of this manual.

EXAMPLES: x=ANALOG 'Sets variable x to the analog input voltage.

AND

Boolean Operator

ACTION:

PROGRAM SYNTAX:

REMARKS:

The logical AND operator is used in boolean expressions.

expression1 AND expression2

The AND operator uses this "truth table":

expression1	expression2	Condition result
True	True	True
True	False	False
False	True	False
False	False	False

The result is true if both expressions are true.

EXAMPLES:

if (x >2 AND y < 3) then goto INDEX
'The controller checks to see if x > 2 **and** y < 3. If both conditions are true the program goes to a label called INDEX.

ASC

String Manipulation

ACTION:

Returns the ASCII code for the first character in a string.

PROGRAM SYNTAX:

ASC(n$)

REMARKS:

The ASCII code returned is for the first character in the string variable n$. If the string is a null string then a 0 will be returned.

EXAMPLES:

```
INTEGER     x
STRING      a$
a$="part#"
x=ASC(a$)     ' sets x=112     'p'
```

BCD

I/O Operator

ACTION: Returns the value on the BCD switches.

PROGRAM SYNTAX: BCD - used in an expression

REMARKS: BCD
Evaluates and returns the BCD switches as a signed Integer value. The BCD switches are restricted to seven digits with a sign.

Note: The use of the **BCD** command takes precedent over the OUT command and will toggle OUT3-OUT6 when called to strobe the BCD switch bank. Subsequent to a BCD call, an OUT(3)-OUT(6) command **will also set the output** to the appropriate state. If OUT3-OUT6 are used as general purpose outputs, care must be taken not to invoke a BCD command or the state of the outputs will be disturbed.

EXAMPLES:

```
a=BCD          'returns the BCD switches value to variable "a".
```

BOOST

Motion Parameter

ACTION: Enables or disables the Boost Current feature of a stepper or returns the commanded boost status.

PROGRAM SYNTAX: BOOST=expression
BOOST - used in an expression

REMARKS: BOOST=expression
If the expression is true (non-zero) then the BOOST feature is enabled, the current will be boosted to the project configuration boost percentage setting during motion. If the expression is false (zero) then the BOOST feature is disabled, the current will be the normal current during motion.

BOOST - used in an expression
Returns the current setting of the BOOST feature.

EXAMPLES:

```
BOOST=1      ' enables the BOOST feature during motion

BOOST=0      ' disables the BOOST feature during motion
```

```
X=BOOST      ‘ returns the current setting for the BOOST feature
```

BUSY

Motion

ACTION: Returns the motion status.

PROGRAM SYNTAX: BUSY - used in an expression

REMARKS: If the commanded motion is incomplete, BUSY returns a true (1) otherwise BUSY returns a false (0).

EXAMPLES:

```
DO WHILE BUSY              'prints system absolute position
        PRINT#1,ABSPOS     'while motion is still occurring
LOOP
```

CHR$

String Manipulation

ACTION: Returns a one character string whose ASCII code is the argument.

PROGRAM SYNTAX: CHR$(code)

REMARKS: CHR$ is commonly used to send a special character to the serial port.

EXAMPLES:

```
PRINT#1,"Input Accel",CHR$(27)      ' transmits "Input Accel" <ESC> to the host serial port.
```

DECEL

ACTION: Sets or returns the deceleration rate of the axis.

PROGRAM SYNTAX:

DECEL=expression
DECEL - used in an expression

REMARKS:

DECEL=expression
Sets the deceleration rate value in units/sec^2.

DECEL - used in an expression
Evaluates and returns the present deceleration value.

The rate at which the motor speed is decreased. Specifying a 0 or negative value will result in error code 7. Specifying a value greater than **Max Accel**, set in the *Configuration and Setup*, will result in DECEL being set to the **Max Accel** value. At power up and each time a program is run DECEL is initialized to 50% of **Max Accel** value. DECEL can be set during motion, but the new setting will not be used until the next move. Reading DECEL returns the most recent setting. The lowest allowable deceleration rate is 0.07276 rev/sec^2. The highest highest allowable deceleration rate is 1192 rev/sec^2.

EXAMPLES:

DECEL=3.1
sets the deceleration value to 3.1 units/sec^2.

X = DECEL
Sets variable X equal to the value of deceleration.

DEFINE

Miscellaneous Command

ACTION:

Defines a symbolic name to be a particular string of characters.

PROGRAM SYNTAX:

#DEFINE name@1, ... , @10 replacement text
#DEFINE name replacement text

REMARKS:

The name has the same form as a variable name: a sequence of letters and digits that begins with a letter. The name is case sensitive. Typically upper case is used for the name.

The @1, ... , @10 are the program command substitution arguments for the replacement text.

The replacement text can be any sequence of letters of characters.

Any occurrence of the name in the program, not in quotes and not part of another name, will be replaced by the corresponding replacement text when the program is compiled.

EXAMPLES:

#DEFINE TRUE 1
Substitutes a 1 when the name TRUE is encountered.

#DEFINE FALSE 0
Substitutes a 0 when the name FALSE is encountered.

#DEFINE SENDPOS @1 PRINT#@1,ABSPOS
Sends the absolute position via port @1.

SENDPOS 1
Sends the absolute position via port #1. The 1 is substituted for the @1 .

```
#DEFINE CLR   PRINT#2,CHR$(12);
#DEFINE LOCATE @1,@2     PRINT#@,CHR$(27);"[@1,@2H";

CLR                ' clear display
LOCATE 1,2         ' locate cursor at row 1 column 2
```

DIST

Trajectory Parameters

ACTION:

Returns the distance moved from the start of the last commanded motion or changes the move distance during indexed motion.

PROGRAM SYNTAX:

DIST = expression
DIST - used in an expression

REMARKS:

DIST = expression
Extends or shortens the index (MOVEI or MOVEA) motion underway. A positive value extends the move, a negative value shortens it. If the present move is past the point to which the move has been shortened, by a DIST = negative value, then the move is stopped. The DIST command has no effect if the present move is currently stopping.

DIST - used in an expression
Returns the distance traveled from the start of the last motion command. DIST returns a positive number, regardless of the move direction.

EXAMPLES:

x=DIST
sets x to the distance moved from the start of motion.

```
MOVEI = -25
DIST = -10          'shortens the move by 10 units
```

DO...LOOP

ACTION:

Repeats a block of statements **while** a condition is true or **until** a condition becomes true.

PROGRAM SYNTAX 1:

```
DO {UNTIL | WHILE} [condition]
[statement block]
[EXIT DO]
[statement block]
LOOP
```

PROGRAM SYNTAX 2:

```
DO
[statement block]
[EXIT DO]
[statement block]
LOOP {UNTIL | WHILE} [condition]
```

REMARKS:

Syntax 1 allows the condition to be tested at the top of the loop. Syntax 2 allows the condition to be tested at the bottom of the loop therefore the loop will always execute at least once.

EXIT DO is an alternative exit from a DO...LOOP.

EXIT DO transfers control to the statement following the LOOP statement. When used within nested DO...LOOP statements, EXIT DO transfers out of the immediately enclosing loop. EXIT DO can be used only in a DO...LOOP statement.

EXAMPLES:

```
DO WHILE EVENT1(1) <> 1
        'Continue the loop while event1 does not equal 1.
        statements
LOOP                                    'End of loop.
```

ENCPOS

Trajectory Parameters

ACTION: Returns the encoder position.

PROGRAM SYNTAX: ENCPOS - used in an expression

REMARKS: ENCPOS
Evaluates and returns the present encoder position.

The actual motor position as read from an incremental encoder. The range of ENCPOS is ±2,147,483,647 encoder counts. Reading ENCPOS returns the actual motor position in **user units**. ENCPOS is initialized to 0 at power up. Setting ABSPOS sets ENCPOS to the same value. Drive type in the configuration system folder must be set to closed loop stepper to read ENCPOS. ENCPOS will read zero for open loop stepper systems.

EXAMPLES:

```
y = ENCPOS          ' returns the encoder position to variable y.
```

ENCSPD

Trajectory Parameters

ACTION: Returns the current encoder speed in units/sec.

PROGRAM SYNTAX: ENCSPD - used in an expression

REMARKS: ENCSPD
Evaluates and returns the current encoder speed.
Drive type in the configuration system folder must be set to closed loop stepper to read ENCSPD. ENCSPD will read zero for open loop stepper systems.
Reading ENCSPD returns the actual motor speed with a resolution of:

(122.07 * "Units/rev")/"Line count" units/sec.

These values are set in the *Configuration and Setup*.

Example: "Units/rev" = 1 , "Line count" = 500
example resolution = .24414 units/sec

The returned motor speed value is a signed number.

EXAMPLE:

```
x=ENCSPD          'returns the current encoder speed to variable x.
```

END

ACTION: Signifies the end of a program.

PROGRAM SYNTAX: END

REMARKS: This command signifies the end of a program and must be included in each program or an error condition may occur.

EXAMPLES:

```
statement
....
END
```

ERR

Return Error Code

ACTION: Returns the error status of the controller.

PROGRAM SYNTAX: ERR - used in an expression

REMARKS: If an error occurs while the program is running, it jumps to label ERROR_HANDLER, if present, otherwise it ends. The fault LED is on while the error code is non-zero. Host command "ERR" or executing a "GOTO" command in the error handler code clears the error code. The first error locks out subsequent errors.

Error Code	Description
0	No error.
1	Could not burn flash successfully.
2	Could not download file.
3	Not enough memory to execute user program.
4	Attempt to access a non-existent array element.
5	Real data too large to convert to Integer data.
6	Attempt to set accel data <= 0.
7	Attempt to set decel data <= 0.
8	Attempt to access non-existent output.
9	Attempt to access non-existent input.
10	Attempt to divide by 0.
11	Received serial data will not fit in buffer.
12	Motion occurring when program ended.
13	Attempt to execute user program that is not present.
14	Incorrect user program checksum.
15	reserved for future use.
16	reserved for future use.
17	reserved for future use.
18	reserved for future use.
19	Attempt to set FOLERR <0.
20	reserved for future use.
21	Move distance too large.
22	Function not implemented.
23	INPUT command error occurred
128	+limit switch activated.
129	-limit switch activated.
130	+ Software travel limit exceeded.
131	- Software travel limit exceeded.
132	reserved for future use.
133	Excessive position error.

134	Registration distance too small.
135	Attempt to move with drive not enabled.
136	Attempt to move with drive not ready.
137	Closed loop correction failure.

If there an error handler routine is not present in the program, then an error will simply terminate program execution, otherwise an error causes the program to jump to the error handler routine (label **ERROR_HANDLER**). The error handler routine can not be interrupted. The error handler routine is terminated with either an END statement or a GOTO <label> statement. The END statement will terminate program execution. The GOTO <label> statement will cause program execution to continue with the line labeled <label>. At this point the program can be interrupted.

EXAMPLES:

```
x=ERR
'Sets x equal to the present controller error number for this task and clear the error number.
```

EVENT1

Motion

ACTION:

Sets the trigger polarity and trigger enable, which are used in a MOVEHOME and MOVEREG cycle.

PROGRAM SYNTAX:

EVENT1=expression

REMARKS:

The EVENT1 command is used to select the effect of the hardware signal at the EVENT1 / IN1 input. This input is typically wired to a switch or sensor. It may be used as a home position trigger during a MOVEHOME cycle. It also may be used as a position mark registration trigger during a MOVEREG cycle. When used for mark registration, a trigger on EVENT1 will initiate the index portion of the MOVEREG cycle.

The EVENT1 triggering for a MOVEHOME or MOVEREG cycle may be combined with an encoder index pulse input, and is assigned in the user program *Configuration and Setup*.

For a **MOVEHOME** cycle, the EVENT1 command may be used to set the polarity of the move home trigger. If the expression to the right of the EVENT1 command is **positive**, for example EVENT1 = 1, the home cycle trigger occurs when the EVENT1 input becomes active. If the expression to the right of the EVENT1 command is **negative**, for example EVENT1 = -1, the home cycle trigger occurs when the EVENT1 input becomes inactive. An EVENT1 home trigger cannot be disabled using this command.

For a **MOVEREG** cycle, the EVENT1 command may be used to set the polarity of the registration trigger. If the expression to the right of the EVENT1 command is **positive**, for example EVENT1 = 1, the registration cycle trigger occurs when the EVENT1 input becomes active. If the expression to the right of the EVENT1 command is **negative**, for example EVENT1 = -1, the registration cycle trigger occurs when the EVENT1 input becomes inactive.

The EVENT1 trigger for a registration cycle may be **disabled** by setting EVENT1=0. A registration trigger may be enabled to either polarity during a move. It may not, however, be disabled once the cycle has begun.

The EVENT 1 input state can be read with command IN(1).

EXAMPLES:

EVENT1=0 disables EVENT1 trigger if assigned as a MOVEREG trigger.

EVENT1=1 Sets EVENT1 trigger to positive polarity triggering and enables the trigger.

EVENT1=-1 Sets EVENT1 trigger to negative edge triggering and enables the

trigger.

EVENT2

Motion

ACTION:

Sets the trigger polarity and trigger enable, which are used in a MOVEHOME and MOVEREG cycle.

PROGRAM SYNTAX:

EVENT2=expression

REMARKS:

The EVENT2 command is used to select the effect of the hardware signal at the EVENT2 / IN2 input. This input is typically wired to a switch or sensor. It may be used as a home position trigger during a MOVEHOME cycle. It also may be used as a position mark registration trigger during a MOVEREG cycle . When used for mark registration, a trigger on EVENT2 will initiate the index portion of the MOVEREG cycle.

The EVENT2 triggering for a MOVEHOME or MOVEREG cycle is assigned in the user program *Configuration and Setup*.

For a **MOVEHOME** cycle, the EVENT2 command may be used to set the polarity of the move home trigger. If the expression to the right of the EVENT2 command is **positive**, for example EVENT2 = 1, the home cycle trigger occurs when the EVENT2 input becomes active. If the expression to the right of the EVENT2 command is **negative**, for example EVENT2 = -1, the home cycle trigger occurs when the EVENT2 input becomes inactive. An EVENT2 home trigger cannot be disabled using this command.

For a **MOVEREG** cycle, the EVENT2 command may be used to set the polarity of the registration trigger. If the expression to the right of the EVENT2 command is **positive**, for example EVENT2 = 1, the registration cycle trigger occurs when the EVENT2 input becomes active. If the expression to the right of the EVENT2 command is **negative**, for example EVENT2 = -1, the registration cycle trigger occurs when the EVENT2 input becomes inactive.

The EVENT2 trigger for a registration cycle may be **disabled** by setting EVENT2=0. A registration trigger may be enabled to either polarity a move. It may not, however, be disabled once the cycle has begun.

The EVENT 2 input state can be read with command IN(2).

EXAMPLES:

EVENT2=0 disables EVENT2 trigger if assigned as a MOVEREG trigger.

EVENT2=1 Sets EVENT2 trigger to positive edge triggering and enables the trigger.

EVENT2=-1 Sets EVENT2 trigger to negative edge triggering and enables the trigger.

FOLERR

Trajectory Parameter

ACTION:

Sets or returns the position error limit.

PROGRAM SYNTAX:

FOLERR=expression
FOLERR - used in an expression

REMARKS:

FOLERR= expression
Sets the position error limit in units. Setting the position error limit to zero sets the position error limit to 32767 encoder counts.

FOLERR - used in expression
Returns the value of the position error limit.

FOLERR sets or reads the "Following Error" Limit. "Following Error" is the absolute value of the difference between the commanded and actual motor position, i.e. |ABSPOS - ENCPOS|. The test for excessive "Following Error" is only performed by a closed loop stepper whenever the error action in the *Configuration and Setup* is not set to disabled. If the "Following Error" exceeds the FOLERR setting, the action taken is dependent upon the error action selected. If the error action in the *Configuration and Setup* is set to **stop on error** the error code is set to 133 and any motion taking place is terminated. With error action in the *Configuration and Setup* is set to **correct on error**, the motor will attempt to correct the error by making a new move to the required position. This move will be attempted for as many times as necessary up to the number of correction attempts specified in the configuration. When set to **restart on error**, the entire move is restarted after the time between attempts has elapsed. FOLERR is limited to the number of user units corresponding to 32767 encoder counts. FOLERR is initialized to the number of units corresponding to .05 revolutions at power up and each time a project is run. A negative setting for FOLERR results in error code 19. If an attempt is made to set FOLERR greater than 32767 encoder counts, the FOLERR is set to its maximum value of 32767. Reading FOLERR returns the present setting in user units.

EXAMPLES:

```
FOLERR=.5          ' position error limit is set to .5 units
x=FOLERR           ' returns the current position error limit.
```

FOR...NEXT

ACTION:

Repeats a block of statements a specified number of times.

PROGRAM SYNTAX:

FOR *counter = start#* TO *end#*
[statement block]
[EXIT FOR]
[statement block]
NEXT counter

REMARKS:

Counter is a variable used as the loop counter.
Start# is the initial value of the counter.
End# is the ending value of the counter.

The step size is always 1.

If *start* is greater than *end* then the loop will not execute, control is transferred to the statement following the NEXT statement. If *start* equals *end* then the loop will execute once.

EXIT FOR is an alternative exit from a FOR...NEXT loop.

EXIT FOR transfers control to the statement following the NEXT statement. When used within nested FOR...NEXT statements, EXIT FOR transfers out of the immediately enclosing loop. EXIT FOR can be used only in a FOR...NEXT statement.

EXAMPLES:

```
for x=1 to 8            ' For..next loop initialization
        statements      ' Program statements.
next x                  ' End of loop.
```

GETCHAR

ACTION:

Waits for a character on the selected serial port and returns the ASCII code of the character.

PROGRAM SYNTAX:

GETCHAR(n) - used in an expression

REMARKS:

The n specifies the serial port number (1 or 2). Port 1 is the Host port and Port 2 is the User port.

Program execution is suspended while GETCHAR waits for a character to be received by the designated serial port. If a character is already in the receiver buffer the ASCII code of the character is returned immediately.

EXAMPLES:

```
INTEGER     a,b
STRING      a$,b$
a=GETCHAR(1)        ' sets a to the ASCII code of host character
b=GETCHAR(2)        ' sets b to the ASCII code of user character
a$=a$ + CHR$(A)     ' add host character to a$
b$=b$ + CHR$(A)     ' add host character to b$
```

GOSUB... RETURN

ACTION:

Branches to, and returns from, a subroutine.

PROGRAM SYNTAX:

GOSUB [linelabel]

REMARKS:

You can call a subroutine any number of times in a program. You can call a subroutine from within another subroutine (nesting).

Subroutines can only be nested ten deep.

The execution of the RETURN statement causes the subroutine to goto the line following the call or jump to the subroutine..

Subroutines can appear anywhere in the program, but it is good programming practice to make them readily distinguishable from the main program.

EXAMPLES:

```
GOSUB GET_CHAR          'goto subroutine at label "GET_CHAR"
MOVEI=10                ‘Line that executes after the return.
:
                        :
GET_CHAR:               'label for subroutine
:
        statement block 'statements to perform action of the subroutine
:
RETURN                  'return to program line following GOSUB
                        GET_CHAR
```

GOTO

ACTION: Branches unconditionally to the specified label.

PROGRAM SYNTAX: GOTO [label]

REMARKS: The GOTO statement provides a means for branching **unconditionally** to another label.

It is good programming practice to use subroutines or structured control statements (DO... UNTIL, FOR...NEXT, IF..THEN...ELSE) instead of GOTO statements, because a program with many GOTO statements can be difficult to read and debug. **Try to avoid using "GOTO"!**

EXAMPLES:

```
if x=1 then GOTO coolant_off
:
:
coolant_off:
(statements)
```

HARDLIM OFF

Over Travel Limit

ACTION: Disables the hardware limit inputs.

PROGRAM SYNTAX: HARDLIMOFF

REMARKS: Hard limit inputs are used to stop the motor before it runs into a physical end of travel, thus avoiding damage to the mechanical system. A separate hard limit input is provided for + and - motor rotation. Activating the + input stops the motor if it is rotating in the + direction. Activating the - input stops the motor if it is rotating in the - direction.

Inputs 3 and 4 become general purpose inputs with this command.

EXAMPLES:

```
HARDLIMOFF          ' hard limit inputs are general purpose.
```

HARDLIMON

Over Travel Limit

ACTION:

Enables the hardware limit inputs.

PROGRAM SYNTAX:

HARDLIMON

REMARKS:

Hard limit inputs are used to stop the motor before it runs into a physical end of travel, thus avoiding damage to the mechanical system. A separate hard limit input is provided for + and - motor rotation. Activating the + input stops the motor if it is rotating in the + direction. Activating the - input stops the motor if it is rotating in the - direction.

Inputs 3 and 4 become the +Limit and -Limit inputs. As hard limits, the active signal level can also be configured as active on switch closing or active on switch opening. This is done in the project's *Configuration and Setup*.

The +Limit is only checked when motion in the + direction is commanded, likewise the -Limit is only checked when motion in the - direction is commanded. When a Limit input is activated, the motor is decelerated to a stop using the maximum accel value (set in the project's *Configuration and Setup*) and an error code is set. Code 128 is set when the +Limit is activated and code 129 when the - Limit is activated.

The state of the +Limit can be read with the IN(3) command and the state of the -Limit can be read with the IN(4) command.

EXAMPLES:

```
HARDLIMON          ' Limit inputs are active.
```

HEX$

String Manipulation

ACTION: Returns the hex string of an Integer value.

PROGRAM SYNTAX: A$=HEX$(expression)

REMARKS: The expression must be an integer value.

EXAMPLES:

```
A$=HEX$(255)     ' returns the string "FF"
```

HVAL

String Manipulation

ACTION: Returns the decimal value of a hexadecimal string.

PROGRAM SYNTAX: x=HVAL(A$)

REMARKS: A$ is the designated string variable or string literal.

The converted value is an Integer. Thus "x" must be defined as an Integer.

EXAMPLES:

```
x=HVAL("0XFF")              ' x is set to 255
A$="1F"
x=HVAL(A$)"           ' x is set to 31
```

IF...THEN... ELSE... ENDIF

ACTION:

Allows conditional execution based on the evaluation of a Boolean condition.

PROGRAM SYNTAX 1:

IF condition THEN thenpart [ELSE elsepart]

PROGRAM SYNTAX 2:

```
IF condition1 THEN
[statement block-1]
[ELSE]                  ELSE and statement block-2 is optional
[statement block-2]]
END IF
```

REMARKS:

The argument condition is an expression that SEBASIC evaluates as true (nonzero) or false (zero).

The argument statement block includes any number of statements on one or more lines.

The argument thenpart includes the statements or branches performed when condition is true.

The argument elsepart includes the statements or branches performed when condition is false. The syntax is the same as thenpart. If the ELSE clause is not present, control passes to the next statement in the program following the END IF.

EXAMPLES:

```
if x=0 then
        statement block
else
        statement block
end if
```

IN

I/O Operator

ACTION: Returns the state of a digital input.

PROGRAM SYNTAX: IN(nn) - used in an expression

REMARKS: nn is the specified digital input 1-17.

The value returned is 1 for active or 0 for inactive.

The inputs are assigned as follows:

Input	Signal Designation
1	Event1 / In1
2	Event2 / In2
3	"+Limit" / In3
4	"-Limit" / In4
5	"Run" / In5
6	"Clear" / In6
7	"Feedhold" / In7
8	In 8
9	BCD0 / In 9
10	BCD1 / In 10
11	BCD2 / In 11
12	BCD3 / In 12
13	BCD4 / In 13
14	BCD5 / In 14
15	BCD6 / In 15
16	BCD7 / In 16
17	Drv Ready

Inputs 3 through 7 are individually selectable in the *Configuration and Setup* as either dedicated or general purpose inputs. If selected as dedicated inputs and activated, these inputs cause specific action to occur as outlined in the HARDWARE INPUTS section of this manual.

Inputs selected as general purpose may be used within the user program as needed. A general purpose input will **not** cause the dedicated action to occur when the input is active. Note: The IN(X) command will return the value at the input pin **regardless of the *Configuration and Setup***. For example, if Input 7 is selected in the system configuration as dedicated to "Feedhold", and the input at the pin is active, then IN(7) will return a 1.

EXAMPLES: IF IN(6)=1 then goto continue

INCHAR

ACTION:

PROGRAM SYNTAX:

REMARKS:

Returns the ASCII code of a character from the designated serial port. If no character is in the receiver buffer a 0 is returned.

INCHAR(n)

The n specifies the serial port (1 or 2). Port 1 is the Host port and Port 2 is the User port.

If **no character** has been **received** by the designated serial port**,** a **0 is returned**. Otherwise, the ASCII code value equivalent is returned.

EXAMPLES:

```
INTEGER     x
STRING      a$
DO
  x=INCHAR(1)  ' x= character received or a 0
LOOP UNTIL x > 0          ' wait for character
a$=a$+CHR$(x)             'adds input character to a$
```

INCLUDE

Miscellaneous Command

ACTION: Includes a file name with define statements in a user task.

PROGRAM SYNTAX: #INCLUDE drive:\subdir\...\subdir\filename.inc

REMARKS: Drive is the root directory of the drive.

Subdir is the path required to find the file.

Filename is the include filename with extension .inc.

The include file must be a series of #DEFINE statements only and can be used in any project task file.

The iws.inc file is included in the MX2000 software for Windows. This file can be used to control a Superior Electric IWS-120-SE or IWS-30-SE interface panel.

EXAMPLES: #INCLUDE c:\mx2000\iws.inc ‘ include file iws.inc

INPUT

ACTION:

Reads a Line of data from the designated serial port into a string variable.

PROGRAM SYNTAX:

INPUT#1,n$
INPUT#1,n$,var1$[,var2$] ... [var_n$]
INPUT#1,x
INPUT#2,n$
INPUT#2,n$,var1$[,var2$] ... [var_n$]
INPUT#2,x

REMARKS:

This command accepts input characters until a carriage return or linefeed is received by the designated port.

Multiple arguments can be entered on one input line and are separated by a ",". The input arguments can be strings, Integer values and Real values.

INPUT#1 designated the Host port and INPUT#2 designates the User port as the serial receiver port.

EXAMPLES:

The following data was entered via user port: "A555555,100,10.5,20 " cr

Program:

```
string          a$
integer  x,acc
real            y
INPUT#2,a$,x,y,acc    ' sets a$="A555555"
                      ‘ sets x=100
                      ' sets y=10.5
                      ' sets acc=20 units/sec2
```

INSTR

ACTION: Returns the character position of the first occupance of a specified string in another string.

PROGRAM SYNTAX: INSTR(string1$,string2$) - used in an expression

REMARKS: The expression must be an integer variable.

The comparison is case sensitive and returns a 0 if no match is found.

EXAMPLES:

```
a$= "WE part# 215629"
a=INSTR(A$,"Part#")  'returns the starting position of 'part#" in a$;
                        in this case, the value of 4 is returned.
```

INTROFFn

Interrupt

ACTION: Disables an interrupt for an ON...INTRn command.

PROGRAM SYNTAX: INTROFFn

REMARKS: An interrupt causes the program to stop what it is doing, go do something else and then resume from where it was interrupted. There can be up to 4 software interrupts in a program. The conditions that cause the interrupt can be programmed and the interrupts can be enabled or disabled individually. At the start of program execution the interrupts are disabled and must be enabled within the program. They can also be disabled within the program.

The n (1-4) defines the interrupt number to be disabled.

EXAMPLES:

```
INTROFF1        'disables interrupt 1 for an ON...INTRn command.
```

INTRONn

Interrupt

ACTION: Enables an interrupt for an ON...INTRn command.

PROGRAM SYNTAX: INTRONn

REMARKS: An interrupt causes the program to stop what it is doing, go do something else and then resume from where it was interrupted. There can be up to 4 software interrupts in a program. The conditions that cause the interrupt can be programmed and the interrupts can be enabled or disabled individually. At the start of program execution the interrupts are disabled and must be enabled within the program. They can also be disabled within the program.

The n (1-4) defines which ON...INTRn command is enabled.

EXAMPLES:

```
INTRON1     'enables interrupt 1 for an ON...INTRn command.
```

JOG

Motion

ACTION: Jog the motor in a specified direction.

PROGRAM SYNTAX: JOG = expression

REMARKS: JOG=expression
The sign of the expression determines the direction of motion. If the expression is positive or 0, jogging will take place in the positive direction.
If the expression is negative, jogging will take place in the negative direction. The speed of the jog move is determined by the last SPEED command.

Use the STOP command for stopping the motor.

LCASE$

String Manipulation

ACTION: Converts and returns a string with lower case letters.

PROGRAM SYNTAX: string1$=LCASE$(string2$)

REMARKS: **string2$** is copied and all upper case letters are converted to lower case letters and the resulting string is returned **string1$**.

This command is useful for making the INSTR command case insensitive.

EXAMPLES:

```
a$="HELLO"
b$=LCASE$(a$)        ' sets b$="hello"
```

LEFT$

String Manipulation

ACTION: Returns the leftmost characters of a string.

PROGRAM SYNTAX: string2$=LEFT$(string1$,n)

REMARKS: The n is the number of leftmost characters to return. If n is greater than the length of string1$ then the entire string is returned to string2$.

EXAMPLES:

```
b$="Hello World"
a$=LEFT$(b$,7)          ' sets a$= "Hello W"
```

LEN

String Manipulation

ACTION: Return the number of characters in the designated string.

PROGRAM SYNTAX: LEN(string$) - used in an expression

REMARKS: The expression should be an integer type. If the input string is a null string returns a 0.

EXAMPLES:

```
A=LEN("ABCD")                ' sets A=4
```

LOWSPD

ACTION: Sets or returns the starting speed value of a stepping motor.

PROGRAM SYNTAX:

LOWSPD=expression
LOWSPD - used in an expression

REMARKS:

LOWSPD=expression
Sets the starting speed in units/sec. LOWSPD should be set to a positive number or 0. The default is 0.

LOWSPD - used in an expression
Evaluates and returns the present starting speed value.

EXAMPLES:

```
LOWSPD=2.5          'Sets the starting speed to 2.5 units/sec

X=LOWSPD            'Sets X equal to the present starting speed
```


MID$

ACTION: Returns the designated middle number of characters of a string.

PROGRAM SYNTAX:

string1$=MID$(string2$,start,number)

REMARKS:

The **start** specifies the starting position of the input string (string2$).

The **number** specifies the number of characters to return. If the number is greater than the (length of the string - start position) the input string is copied from the starting position to the end of the string.

EXAMPLES:

```
a$="P/N 123AC"
b$=MID$(a$,5,3)      ' sets b$="123"
c$=MID$(a$,5,9)      ' sets c$="123AC"
```

MOVEA

Motion

ACTION: Initiates the motor to move to the specified absolute position.

PROGRAM SYNTAX: MOVEA=expression

REMARKS: The expression represents the specified absolute position.

Move to the specified position. The specified position must not be further than +/- 2,147,483,647 microsteps away or error code 21 will be set and no motion will occur.

EXAMPLES:

```
MOVEA= -1.0          ' moves to an absolute position of  -1.0 units.
```

MOVE HOME

ACTION:

Runs the motor until the home input is activated, captures and records the position of the switch activation as home (electrical zero), then decelerates the motor to a stop.

PROGRAM SYNTAX:

MOVEHOME=expression

REMARKS:

The sign of the expression determines the direction (positive or negative) of motion for the home cycle. The non-zero value of the number is not significant.. The commanded speed is determined by the last SPEED command that was executed.

The MOVEHOME trigger can be the EVENT 1 input, EVENT 2 input or an Encoder marker state. This trigger is defined by the user program *Configuration and Setup*, and also by the EVENT1 or EVENT2 command if they have been executed prior to the MOVEHOME.

Prior to starting a MOVEHOME motion, the appropriate trigger input (EVENT 1 or EVENT 2) is checked to see if it has already been triggered. If the trigger is already triggered the ABSPOS and ENCPOS are set to zero and no motion occurs. Otherwise, the motor accelerates at the ACCEL rate to the commanded SPEED and continues at this speed until the home trigger condition is met. When the home trigger occurs, the motor decelerates to a stop at the DECEL rate. Once at a stop, the distance traveled from the trigger becomes the new ABSPOS and ENCPOS value. The exact position that the motor was at when the trigger occurred becomes the **zero position**, or home.

EXAMPLES:

```
MOVEHOME= -1.0        'Initiates a mechanical home cycle in the
                      negative direction.

MOVEA=0               'Moves motor back to electrical home. (i.e. switch
                      edge)
```

MOVEI

Motion

ACTION: Initiate an incremental move.

PROGRAM SYNTAX: MOVEI=expression

REMARKS: The expression represents the distance to move from its present location. The sign of the expression determines the direction (positive or negative) of motion for the move.

Move the specified incremental distance from the present position. The increment must not be greater than +/- 2,147,483,647 microsteps or error code 21 will be set and no motion will occur.

EXAMPLES:

```
MOVEI= -1.0          ' moves  -1.0 units.
```

MOVEREG

ACTION: Runs the motor until the mark registration input is activated; then moves the motor the desired registration distance.

PROGRAM SYNTAX: MOVEREG=expression

REMARKS: The expression represents the incremental distance to move after a registration trigger has occurred. The sign of the expression determines the direction (positive or negative) of motion for the registration cycle. The distance must not be greater than +/- 2,147,488,647 encoder counts or error code 21 will be set and no motion will occur.

The registration trigger can be the EVENT 1 input, EVENT 2 input or an Encoder marker state. This trigger is defined in the user program *Configuration and Setup*, and also by the EVENT1 or EVENT2 command if they have been executed prior to the MOVEREG.

The Registration Travel Limit, which is set by command **REGLIMIT**, limits the distance that the motor will rotate if no trigger occurs. A **REGLIMIT** setting of 0, sets no limit for motor rotation while awaiting a trigger. This is the condition after power up or RESET. The motor speed during a **MOVEREG** move is set by the **SPEED** command. When the registration trigger occurs, the registration distance is checked to determine if the motion can be stopped in the given distance. If it can't, then the motion will be stopped using the project's *Configuration and Setup* setting for max. accel, and an error code 134 is set. This error can be eliminated by increasing the reg. distance, decreasing the speed or increasing the deceleration.

$$SPEED = \sqrt{MOVEREG * 2 * DECEL * .96}$$

Prior to starting a MOVEREG motion the appropriate trigger input (EVENT 1 or EVENT 2) is checked to see if it has already been triggered. If the trigger has already occurred, an incremental move of the distance specified by the expression to the right of the MOVEREG will occur.

A **MOVEREG** can be started with its trigger disabled (except for the two encoder index marker selections). The registration trigger may then be enabled later by an EVENT1 or EVENT2 command.

EXAMPLES:

```
MOVEREG= 1.0        ' Initiates a positive registration cycle of 1 unit.
```

NOT

Boolean Operator

ACTION:

PROGRAM SYNTAX: The logical NOT operator is used in boolean expressions.

REMARKS: NOT expression

The NOT operator uses the "truth table":
The result is TRUE if the expression is FALSE

expression	condition result
True	False
False	True

EXAMPLES:

```
DO
      :
      :
LOOP UNTIL (NOT (BUSY))      'The controller will continue to execute until
                               the axis is done with motion.
```

ON...INTRn

Interrupt

ACTION: Sets condition to execute subroutine INTRn.

PROGRAM SYNTAX: ON [condition] INTRn

REMARKS: The "n" specifies the interrupt number 1-4.

When the specified condition for the ON...INTRn command becomes TRUE during program execution and the designated interrupt "n" has been enabled, a subroutine call to label INTRn takes place. Upon completion of the subroutine the program continues from where it was interrupted and execute the next program line. The INTRn can be disabled at any time during program execution by the INTROFFn command. The INTRn can be enabled at any time during program execution by the INTRONn command.

The <condition> for each enabled interrupt is checked at the end of execution of **each** program line. The first <condition> that is TRUE will cause the interrupt to occur. Because the operating system must check all <conditions> for enabled interrupts after every program line, excessive use of software interrupts will slow down the execution of the user's program.

The following example shows the execution flow for two conditions to be tested. The first condition which is true will result in execution of the appropriate interrupt routine, in this case INTR1: or INTR2:

ON <condition1> **INTR1**
ON <condition2> **INTR2**

INTRON1	'turn on interrupt 1
program checks <condition 1>	'check condition 1, if it's TRUE jump to code at INTR1: if not, continue with next program statement
PROGRAM STATEMENT	'execute normal program line
program checks <condition 1>	'check condition 1, if it's TRUE jump to code at INTR1: if not, continue with next program statement
INTRON2	'turn on interrupt2
program checks <condition 1>	'check condition 1, if it's TRUE jump to code at INTR1: If not, continue
program checks <condition 2>	'check condition 2, if it's TRUE jump to INTR2: If not execute next program statement.

NEXT PROGRAM STATEMENT program checks <condition 1>	'execute next line in program 'check condition 1, if it's TRUE jump to code at INTR1: If not, continue and program checks <condition 2>, 'check condition 2, if it's TRUE jump to INTR2: If not execute next program line.
INTROFF1 program checks <condition 2>	'check condition 2, if it's TRUE jump to INTR2: If not execute next program line.
INTROFF2	'condition 2 is not checked since Interrupt 2 has been disabled.
NEXT PROGRAM STATEMENT	'execute next line in program no conditions are checked since both interrupt 1 and interrupt 2 were disabled with the INTROFF command.
INTR1: **PROGRAM STATEMENTS**	'beginning of interrupt 1 routine 'execute program statement interrupt conditions are not checked after program statements within the interrupt routine.
RETURN	'end of interrupt 1 routine
INTR2: **PROGRAM STATEMENTS**	'beginning of interrupt 2 routine 'execute interrupt 1 routine statement interrupt conditions are not checked after program statements within the interrupt routine.
RETURN	'end of interrupt 1 routine

Up to four interrupt subroutines can be embedded in the program code. A RETURN command is required at the end of each subroutine. There are four interrupt subroutines labeled (INTR1-INTR4).

EXAMPLES:

```
ON IN (5)=1 INTR1          ' specifies input 5=1  as the condition to go sub
INTR1
program statements
INTRON1                    ' enables INTR1
program statements

INTR1:
program statements
RETURN
```

OR

Boolean Operator

ACTION: The logical OR operator is used in boolean expressions.

PROGRAM SYNTAX: expression1 OR expression2

REMARKS: The OR operator uses this "truth Table":
The result is TRUE, if either expression is TRUE.

Expression1	Expression2	Condition Result
True	True	True
True	False	True
False	True	True
False	False	False

EXAMPLES:

```
DO

LOOP until (A > 5 OR X = 0)
'The controller continues to do the loop Until variable A>5 or variable X=0.
```

OUT

I/O Operator

ACTION: Sets or returns the condition of a specified digital output.

PROGRAM SYNTAX:

OUT(n) = expression
OUT(n) - used in an expression

REMARKS: n is the specified output (1-8).

OUT(n)
Returns a 1 for a commanded active output and a 0 for a commanded inactive output.

OUT(n) = expression
If the expression is a non-zero value the specified output will be activated.

The output are assigned as follows:

Output	Output Designation
1	Out 1
2	Out 2
3	Out 3
4	Out 4
5	BCD Strobe 0 / Out 5
6	BCD Strobe 1 / Out 6
7	BCD Strobe 2 / Out 7
8	BCD Strobe 3 / Out 8

EXAMPLES:

```
OUT(1)=1          ' sets OUT 1 to the active state.
OUT(2)=0          ' sets OUT 2 to the inactive state
x=OUT(1)            Gets the state of output 1 and stores it to x.
```

Note that use of the **BCD** command takes precedent over the **OUT** command and will toggle OUT3-OUT6 when called to strobe the BCD switch bank. If OUT3-OUT6 are used as general purpose outputs, care must be taken not to invoke a BCD command or the state of the outputs will be disturbed.

PRINT

ACTION: Transmits designated data via the designated serial port.

PROGRAM SYNTAX:

PRINT#1,[expression][, or ;][expression][, or ;]
PRINT#2,[expression][, or ;][expression][, or ;]

REMARKS: Port 1 is the Host port and Port 2 is the User Port.

expression can be an integer variable, real variable, parameter, string variable or Literal string. Literal strings must be enclosed in quotation marks.

If a comma "**,**" is used between expressions five spaces will separate expressions.

If a semicolon "**;**" is used between expressions there will be no space between expressions.

Up to 20 expressions can be used with one PRINT command.

If a semicolon "**;**" is used at the end of the PRINT command, no carriage-return/line-feed sequence will be generated.

EXAMPLES:

```
ACCEL=10.5
DECEL=2.1
PRINT#1,"accel= ";ACCEL,"decel= ";DECEL
' Host output "accel= 10.5     decel= 2.1" crlf

ACCEL=10.5
DECEL=2.1
PRINT#2,"accel= ";ACCEL,"decel= ";DECEL
        ' User output "accel= 10.5     decel= 2.1" crlf

ACCEL=10.5
DECEL=2.1
PRINT#2,"accel= ";ACCEL,"decel= ";DECEL;
        ' User output "accel= 10.5     decel= 2.1"
```

PRINT USING

ACTION: Prints strings character or formatted numbers.

PROGRAM SYNTAX:

```
PRINT USING #1,"literal string",[exp][, or;][exp][;]
PRINT USING #1,Format$,[exp][, or;][exp][;]
PRINT USING #2,"literal string",[exp][, or;][exp][;]
PRINT USING #2,Format$,[exp][, or;][exp][;]
```

REMARKS: Port 1 is the Host Port and Port 2 is the User Port.

The numeric values are formatted only using the literal string or a designated Format$ variable string. This string can contain non-format characters that will be printed prior to the formatted number. The following characters in the string will not be printed from the string:
"+" "#" "0" " ." "\" and ",". However, these character can be printable characters by preceding the character with a "\".

Example:

requirement to send the following ASCII string with the current state of OUT(1) (Output #1 is <state> which is the coolant control)

```
a$="Output \#1 is # which is the coolant control"
PRINT USING #1,a$,OUT(1)
```

The resulting serial output:
Output #1 is n which is the coolant control
where: n is the state of output (1)

The "," which is the delimiter for expressions will not print spaces like the PRINT # command. If spaces are required between expressions they must be added to the literal string or format$.

Example:

```
ACCEL=10000
DECEL=20000
a$="Acc= 000000   Dcc= 000000"
PRINT USING#1,a$,accel,decel
```

The resulting serial output:
Acc= 010000 Dcc= 020000

If the numeric data is larger than the specified format than an "*" will be substituted for the 0's and #'s in the output.

Example:

```
ABSPOS=1000.54
a$="Position= +0##.##"
PRINT USING #1,a$,abspos
```

The resulting serial output:

```
Position= +***.**
```

The following special characters are used to format the numeric field:

+	The sign of the number will always be printed. The default prints the negative sign and substitutes a space for the positive sign.
#	Represents a digit position. If no data exists at the digit position substitutes a space. The digit field will always be filled.
0	Represents a digit position. If no data exists at the digit position substitutes a zero. The digit field will always be filled.
.	A decimal point may be inserted at any position in the field.

The valid formats are:

Left side format	**Comments**
+0000	The sign with leading zero's will be printed.
+0000.	The sign with leading zero's and decimal point will be printed. The right side format is optional.
+####	The leading spaces with a sign and digits will be printed.
+####.	The leading spaces with a sign, digits and decimal point will be printed. The right side format is optional.
0000	The - sign or a space with leading zero's will be printed.
0000.	The - sign or a space with leading zero's and decimal point will be printed. The right side format is optional.
####	The leading spaces with a - sign or a space and digits will be printed.
####.	The leading spaces with a - sign or a space, digits and decimal point will be printed. The right side format is optional.
+.	The sign and decimal point will be printed. This requires the right side format also.
.	The - sign or a space and decimal point will be printed. This requires the right side format also.

Right side format	Comment
0000	Prints digits with trailing zero's.
####	Prints digits with trailing spaces.
00##	Prints two digits minimum with trailing spaces.

If the expressions are literal strings or variable strings they will be printed as is.

If a semicolon is used at the end of the Print Using command, no carriage-return / line-feed sequence will be generated.

When numeric data is to be printed, the format string is searched from the beginning for a format character (+0#.). The string data up to this position is sent via the serial port. The format characters (+0#.) are now processed and the formatted value is sent via the serial port. When the next numeric data is to be printed, this process continues from the current position in the string. When the end of the format string is encountered and numeric data is to be printed, a default format (PRINT # format) is used. If the format string end is not encountered and the command is complete the remaining characters in the format string will be printed.

The following example illustrates how the format string is processed. The command is:

PRINT USING#1,"Numbers are +###.## ### 0## **",100.54,"mv", 999,"cnts" ,54," is limit"

The **"Numbers are "** is extracted from the string and sent via serial port. The **"+###.##"** is extracted from the string as the data format, which results in "+100.54" being sent via serial port. The string **"mv"** is sent via serial port. The **" "** is extracted from the string and sent via serial port. The **"###"** is extracted from the string as the data format, which results in "999" being sent via serial port. The string **"cnts"** is sent via serial port. The **" "** is extracted from the string and sent via serial port. The **"0##"** is extracted from the string as the data format, which results in "054" being sent via serial port. The string **" is limit"** is sent via serial port. The **" **"** is extracted from the string and a crlf is appended and sent via serial port. The resulting string is:

Numbers are +100.54mv 999cnts 054 is limit**<cr><lf>

EXAMPLES:

```
PRINT USING #1,"The time is ##,:##am",12,30
        The time is  12: 30am<cr><lf>

PRINT USING #1,"today   s date is 00\\00\\####",1,31,1980;
        Today's date is  01\ 31\ 1980

ABSPOS=10560.32
PRINT USING #1,"Absolute Position is +0######.0## units",abspos
        Absolute Position is +0010560.32  units <cr><lf>
```

REDUCE

Motion

ACTION:

Enables\disables the REDUCE current feature of a stepper or returns the REDUCE current status.

PROGRAM SYNTAX:

REDUCE=expression
REDUCE - used in an expression

REMARKS:

REDUCE=expression
If the expression is true (non-zero) then the REDUCE feature of a stepping motor is enabled. The motor current while motion is stopped will be reduced to percentage selected by the project configuration (0% to 100%) . If the expression is false (zero) then the REDUCE feature of a stepper motor is disabled, the standstill current will be the normal motor current.

REDUCE - used in an expression
Returns the current setting for the REDUCE feature.

EXAMPLES:

```
REDUCE=1    'enables the REDUCE feature of a stepper drive.

REDUCE=0    'disables the REDUCE feature of a stepper drive.

X=REDUCE    'returns the current REDUCE setting to variable X.
```

REGLIMIT

Over Travel Limit

ACTION:

Sets or returns the distance to be moved during a MOVEREG cycle, while awaiting a trigger. If no trigger occurs, a MOVEREG cycle behaves like a MOVEI cycle, with the distance specified by REGLIMIT.
REGLIMIT must be set prior to a MOVEREG cycle.

PROGRAM SYNTAX:

REGLIMIT - used in an expression
REGLIMIT=expression

REMARKS:

REGLIMIT
Return the current MOVEREG travel distance. The value returned is 0.

REGLIMIT=expression

Sets the MOVEREG travel distance. REGLIMIT should be set to a positive number or 0. Setting REGLIMIT = 0, or a negative number, allows a MOVEREG to run indefinitely while awaiting a trigger. If REGLIMIT <0 then REGLIMIT will be set to 0.

EXAMPLES:

```
REGLIMIT= 0 ' disables the MOVEREG travel distance limit.

REGLIMIT= 10          ' set the MOVEREG travel distance limit to 10 units.
```

RIGHT$

String Manipulation

ACTION:

Returns the rightmost characters of a string.

PROGRAM SYNTAX:

string1$=RIGHT$(string2$,n)

REMARKS:

The n is the number of rightmost characters to return. If n is greater than the length of string2$ then the entire string is returned to string1$.

EXAMPLES:

```
b$="Hello World"
a$=RIGHT$(b$,4)       ' sets a$="orld"
```

SOFTLIM NEG

Over Travel Limit

ACTION:

Programmable "software limit switch" for motion in the negative direction. Sets or returns the absolute negative travel limit position value for the motor.

PROGRAM SYNTAX:

SOFTLIMNEG=expression
SOFTLIMNEG - used in an expression

REMARKS:

Software travel limits are used to stop the motor when the commanded position (ABSPOS) exceeds the programmed software travel limit. There are two software travel limits, one for + and one for - motor rotation. The
+ software travel limit is tested when the motor is rotating in the + direction. The - software travel limit is tested when the motor is rotating in the - direction.

The software travel limits are checked if they are enabled and a motion other than **MOVEHOME** is occurring.

The software travel limits power up disabled (**SOFTLIMOFF**). At power up, the -software travel limit is set to **-2,147,481,647 encoder counts** away from 0. This setting is changed with the **SOFTLIMNEG** command.

When a travel limit is exceeded, the motor is decelerated to a stop using the maximum accel value, and an error code is set. Code 130 is set when the + software limit is exceeded and code 131 when the - software limit is exceeded.

SOFTLIMNEG=expression
Sets the absolute travel distance.

SOFTLIMNEG - used in an expression
Evaluates and returns the absolute software travel distance.

EXAMPLES:

```
SOFTLIMNEG = -4   ' Sets the absolute software travel distance to -4 units.
```

SOFTLIM OFF

Over Travel Limit

ACTION: Disables the software over travel limits.

PROGRAM SYNTAX: SOFTLIMOFF

REMARKS: Software travel limits are used to stop the motor when the commanded position (ABSPOS) exceeds the programmed software travel limit. There are two software travel limits, one for + and one for - motor rotation. The + software travel limit is tested when the motor is rotating in the + direction. The - software travel limit is tested when the motor is rotating in the - direction.

This command disables the negative and positive software limits checking during motion.

EXAMPLES:

```
SOFTLIMOFF          'Disables the negative and positive software limits.
```

SOFTLIM ON

Over Travel Limit

ACTION: Enables the software over travel limits.

PROGRAM SYNTAX: SOFTLIMON

REMARKS: Software travel limits are used to stop the motor when the commanded position (ABSPOS) exceeds the programmed software travel limit. There are two software travel limits, one for + and one for - motor rotation. The + software travel limit is tested when the motor is rotating in the + direction. The - software travel limit is tested when the motor is rotating in the - direction.

This command enables the negative and positive software limits checking during motion.

The software travel limits are checked if they are enabled and motion other than MOVEHOME (move to home) is occurring.

The software travel limits are disabled at power up and each time a program is run disabled **(SOFTLIMOFF)** and are set to **2,147,481,647 encoder counts** away from 0. These settings can be subsequently changed with commands **SOFTLIMPOS** and **SOFTLIMNEG**.

When a travel limit is exceeded, the motor is decelerated to a stop using the maximum accel value, and an error code is set. Code 130 is set when the + software limit is exceeded and code 131 when the - software limit is exceeded.

EXAMPLES:

```
SOFTLIMON          'Enables the negative and positive software limits.
```

SOFTLIMPOS

Over Travel Limit

ACTION:

Programmable "software limit switch" for motion in the positive direction. Sets or returns the absolute positive travel limit position value for the motor.

PROGRAM SYNTAX:

SOFTLIMPOS=expression
SOFTLIMPOS - used in an expression

REMARKS:

Software travel limits are used to stop the motor when the commanded position (ABSPOS) exceeds the programmed software travel limit. There are two software travel limits, one for + and one for - motor rotation. The
+ software travel limit is tested when the motor is rotating in the + direction. The - software travel limit is tested when the motor is rotating in the - direction.

The software travel limits are checked if they are enabled and a motion other than **MOVEHOME** is occurring.

The software travel limits power up disabled (**SOFTLIMOFF**). At power up, the +software travel limit is set to +**2,147,481,647 encoder counts** away from 0. This setting is changed with the **SOFTLIMPOS** command.

When a travel limit is exceeded, the motor is decelerated to a stop using the maximum accel value, and an error code is set. Code 130 is set when the + software limit is exceeded and code 131 when the - software limit is exceeded.

SOFTLIMPOS=expression
Sets the absolute travel distance.

SOFTLIMPOS - used in an expression
Evaluates and returns the absolute travel distance.

EXAMPLES:

```
SOFTLIMPOS = +4   ' Sets the absolute travel distance to +4 units.
```

Trajectory Parameters

SPEED

ACTION:

Sets and returns the target velocity of the motor.

PROGRAM SYNTAX:

SPEED = expression
SPEED - used in an expression

REMARKS:

SPEED - used in an expression
Evaluates and returns the target velocity.

Sets the target speed for motion. Specifying a value < 0, results in a target speed of 0. Specifying a value greater than "Max Speed", set in the *Configuration and Setup*, will result in a target speed of "Max Speed". At power up the target speed is initialized to 25% of "Max Speed". SPEED can be set during motion, the new setting is effective immediately.

The lowest programmable speed is 0.00015 rev./second.

EXAMPLES:

```
SPEED=3.0          ' Sets the velocity to 3.0 units/second.
x=SPEED            ' sets x to 3.0.
```

Motion

STOP

ACTION:

Stops any motion with a controlled stop.

PROGRAM SYNTAX:

STOP

REMARKS:

Stop the motor using the programmed decel and velocity profile.

Although any motion in progress will stop, the user program may continue to execute. Therefore, any subsequent move commands will execute as the program continues.

EXAMPLES:

```
STOP               ' generates a motion stop command.
```

STOPERR

Motion

ACTION:

Sets or returns the maximum position error allowed when motion is stopped, referred to herein as "position error band".

PROGRAM SYNTAX:

STOPERR=expression
STOPERR - used in an expression

REMARKS:

STOPERR=expression
Specifies the maximum position error allowed when motion is stopped. Setting the position error band equal to zero will disable any correction attempt at standstill. This value is expressed in Units and is used in position maintenance. The position error band is limited to 32767 encoder counts. At power turn on or when a project is run is set to .005 revolutions.

STOPERR - used in an expression
Returns the current STOPERR value in Units.

EXAMPLES:

```
STOPERR=.1          'sets the position error band to .1 units.

X=STOPERR           'returns the current STOPERR to variable X
```

STR$

String Manipulation

ACTION:

Returns a string representation of a numeric expression.

PROGRAM SYNTAX:

string1$=STR$(numeric_expression)

REMARKS:

The numeric expression can be a parameter value, real value or integer value.

The STR$ command is the complement of a VAL command.

EXAMPLES:

```
STRING      a$,b$,c$
INTEGER     x
REAL        y
ACCEL=10.5
x=100
y=2.1
a$=STR$(ACCEL)    ' sets a$="10.5"
b$=STR$(x)        ' sets b$="100"
c$=STR$(y)        ' sets c$="2.1"
```

String Manipulation

STRING$

ACTION: Returns a string of characters.

PROGRAM SYNTAX: string1$=STRING$(num,code)

REMARKS: The **num** indicates the length of the returned string.

The **code** is the ASCII code of each character.

EXAMPLES:

```
a$=STRING$(10,63)            ' sets a$="??????????"
```

Time Functions

TIMER

ACTION: Sets or returns the timer value.

PROGRAM SYNTAX:

TIMER=expression
TIMER - used in an expression

REMARKS:

TIMER = expression
Sets the timer value to the expression. The value is in seconds.

TIMER
Returns the current timer value to the variable.

The timer is free running and counts **up** in .001 second increments. After reaching a value of +2,147,481.647 seconds, the timer wraps around to -2,147,481.647 and continues to count towards zero (i.e. the next count is -2,147,481.646). Programs which use large timer values must take this into account and adjust appropriately.

EXAMPLES:

```
TIMER=0                  'Sets the Timer value to 0.

DO
        statements
LOOP WHILE TIMER < 1.0           'Do this loop until timer >= 1.0
```

UCASE$

String Manip,Motion

ACTION: Converts and returns a string with upper case letters.

PROGRAM SYNTAX: string1$=UCASE$(string2$)

REMARKS: **string2$** is copied and all lower case letters are converted to upper case letters and the resulting string is returned **string1$**.

This command is useful for making the INSTR command case insensitive.

EXAMPLES:

```
a$="hello"
b$=UCASE$(a$)        ' sets b$="HELLO"
```

UNITID

Daisy Chain

ACTION: Returns the current Unit ID.

PROGRAM SYNTAX: UNITID - used in an expression

REMARKS: The unit id value returned is 1-32. The value read from the unit id switches on power-on.

EXAMPLES:

```
ID =UNITID                  ‘ sets variable ID to the unit id number
IF VAL(unitid$) = ID then   ‘ if received unit id matches the unit id
                              number execute the following statements.
        [statement block]
END IF
```

VAL

String Manipulation

ACTION: Return the numeric value of a string.

PROGRAM SYNTAX: VAL(n$) - used in an expression

REMARKS: **n$** is the designated string.

Only numeric values are returned. The first character that cannot be part of the number terminates the string. If no digits have been processed a value of zero is returned.

EXAMPLES:

```
Integer x
Real    y
STRING  a$,b$
a$="134 Main St"
b$="10.55 dollars"
x=VAL(a$)          ' sets x=134
y=VAL(b$)          ' sets y=10.55
```

WAIT

Time Functions

ACTION: Waits for the specified period of time to expire before continuing.

PROGRAM SYNTAX: WAIT=expression

REMARKS: Program execution is suspended until the desired time has elapsed. The value entered is in seconds.

EXAMPLES:

```
WAIT=1.1          'Waits 1.1 seconds and then continues.
```

WAIT DONE

ACTION:

Waits for a motion to be completed.

PROGRAM SYNTAX:

WAITDONE

REMARKS:

WAITDONE
Waits for motion to be completed before program execution continues.

An alternate way to accomplish the WAITDONE function is as follows:
DO
:
LOOP WHILE BUSY ' Waits until motion is completed.

EXAMPLES:

WAITDONE
'Waits for motion to be complete before continuing program execution .

WNDGS

ACTION:

Enables or disables the stepper drive motor current.

PROGRAM SYNTAX:

WNDGS=expression
WNDGS - used in an expression

REMARKS:

The stepper drive power up with the motor current flowing (enabled) at standstill. The WNDGS command controls the drive enable output at standstill. WNDGS=1 enables the stepping motor, winding current is on when no motion is taking place.WNDGS=0, winding current is off when no motion is taking place.

If the stepper drive is disabled when motion is commanded. The stepper motor windings are turned on and a small delay occurs before motion is started. When the motion is completed a small delay occurs before the windings are turned off again. This delay is programmed in the Configuration **Step Drive Folder**, **Motor current delay** item.

If the stepper motor is configured as a closed loop stepper the position maintenance feature is disabled when the WNDGS command is set to zero. The encoder position is maintained while the drive is disabled. Thus a position error can occur when the WNDGS command is reenabled.

WNDGS=expression
Sets the no motion drive current state. The expression value must be zero or a positive number.

WNDGS - used in an expression
Returns the current setting of the no motion drive current state.

EXAMPLES:

```
WNDGS=1     ' stepper drive current is set to the normal current setting
              with no motion taking place.

WNDGS=0     ' stepper drive current is off  with no motion taking place.

X=WNDGS     ' returns the current  WNDGS state to variable X.
```

Section 6.2

Host Commands Reference Guide

6.2.0 Host Commands

One method of operating the controller is to program it via a PC using the commands detailed in the previous section, then set it up as a stand-alone system. In that case, after it is programmed, the programmer does not need to communicate with any outside computer system. However, another method of system operation involves connecting the controller to some type of "host" computer, via the "HOST RS-232" or "HOST RS-485" port. Then, that computer may direct its operation and query its status from time to time, if desired. To do this, you use the "Host Commands" detailed below. The full command may be spelled out, or the abbreviated commands in parentheses may be used. **Note: Except for the immediate commands, all host commands MUST be preceded by an ESCAPE character for them to be recognized while a program is executing.**

Items placed in quotes (e.g., "?") are key presses or ASCII characters and are not spelled out in letters.

6.2.1 Host Commands Grouped by Function

6.2.2 Host Command Summary (alphabetical list)

Note: The full command may be spelled out, or the abbreviated commands in parentheses may be used.

6.2.3 Host Commands - Alphabetical Listing

Functional list of all HOST commands with syntax and examples.

Notes: "cr" means the carriage return key in the following descriptions. Each command may be spelled out, or the abbreviated command, where applicable, may be used.

<nn

Daisy Chain

ACTION: Enables a specific unit on the Host daisy chain to receive and transmit information.

PROGRAM SYNTAX: <nn cr

REMARKS: "nn" is a unit id number from 01 to 32. Leading zeros are required when specifying unit id numbers < 10. 00 is a special case as described below.

This command is used to communicate to multiple units from a single host computer. In this arrangement the Host communications ports of two or more units are wired together in RS-485 mode as shown previously in the wiring section. Each unit must also have its ID switches set to a unique ID number. One (and only one) unit MUST have its switches set to ID number 1. This unit will transmit a RDY upon reset, the others will not.

In order to accept commands from the Host device, a particular unit must be set to the active mode. The Host accomplishes this by sending the device attention character (<) followed by the two number device ID and a carriage return, line feed. If nn matches the controller ID number as set on the ID switches, that unit becomes the active controller on the chain.

If the Host requires an acknowledgement that a specified unit is in the active mode, the Host may send a <nn? cr . If any unit is on the chain and in the active mode, it will transmit its ID number as two characters.

All controllers on the chain may be placed in a command listen mode. In this mode, all units will actively listen for and respond to commands, but will not transmit any response. This is useful for synchronizing multiple units by simultaneously starting their motion. To place the units in this mode, the Host must send <00 cr. In order to exit the command listen mode an individual unit must be re-activated (e.g. <01).

EXAMPLES:

```
<05        ' sets unit with ID number 5 to the active mode.
<06?       ' queries whether unit 6 is on the chain and active
           ' unit 6 will respond with "06" if it is.
<00        ' sets all units to the command listen mode.
```

ABSPOS

Trajectory Parameters

ACTION: Sets or returns the commanded absolute position of the motor.

PROGRAM SYNTAX:
ABSPOS=number cr
ABSPOS cr

ABBREVIATION: Abbreviation **P** can be used in place of ABSPOS.

REMARKS: See Programming Command **ABSPOS.**

EXAMPLES:

```
ABSPOS=2        'sets absolute position to 2 units.
ABSPOS          'returns the current absolute position
```

ACCEL

Trajectory Parameters

ACTION: Sets or returns the acceleration value of an axis.

PROGRAM SYNTAX:
ACCEL=number cr
ACCEL cr

ABBREVIATION: Abbreviation **AC** can be used in place of ACCEL.

REMARKS: Setting ACCEL less than or equal to zero does not set Error Code 6.

See Programming Command **ACCEL.**

EXAMPLES:

```
ACCEL=2         'Sets the acceleration rate to 2 units/sec^2.
```

ANALOG

I/O Operator

ACTION: Returns the analog input value in volts.

PROGRAM SYNTAX: ANALOG cr

ABBREVIATION: Abbreviation **AN** can be used in place of ANALOG.

REMARKS: See Programming Command **ANALOG.**

EXAMPLES:

```
ANALOG          'returns the analog input voltage value.
```

"BACK SPACE"

Immediate

ACTION: The Backspace key or ASCII code 08 can be used to delete one character from the host receiver buffer.

PROGRAM SYNTAX: Press the BACKSPACE ( ) key or send ASCII 08.

Note: This command is placed in quotes. This is because it is a keypress or ASCII code and is not spelled (typed) out in letters. The ASCII code may be sent to the controller if a keyboard is not used.

BCD

I/O Operator

ACTION: Returns the value on the BCD switches.

PROGRAM SYNTAX: BCD cr

REMARKS: See Programming Command **BCD.**

EXAMPLES:

```
BCD             'returns the BCD switches value.
```

BUSY

Motion

ACTION: Returns the motion status.

PROGRAM SYNTAX: BUSY cr

ABBREVIATION: Abbreviation **BS** can be used in place of BUSY.

REMARKS: See Programming Command **BUSY**

EXAMPLES:

```
BUSY cr         'returns the motion status.
```

"CTRL-A"

Immediate

ACTION:

Stops motion by decelerating the motor using the maximum acceleration value in the *Configuration and Setup*. The system remains energized, and program execution terminates. This command has the same effect as the hardware CLEAR input.

PROGRAM SYNTAX:

Simultaneously press the control key, CTRL, and A keys. ASCII code 01 may also be used.

REMARKS:

"CTRL-A" will stop program execution and motion.

Note: This command is placed in quotes. This is because it is a keypress or ASCII code and is not spelled (typed) out in letters. The ASCII code may be sent to the controller if a keyboard is not used.

"CTRL-C"

Immediate

ACTION:

Performs the same function as the CTRL-A.

PROGRAM SYNTAX:

Simultaneously press the control key, CTRL, and the C keys. ASCII code 03 may also be used.

REMARKS:

"CTRL C" will stop program execution and motion.

Note: This command is placed in quotes. This is because it is a keypress or ASCII code and is not spelled (typed) out in letters. The ASCII code may be sent to the controller if a keyboard is not used.

CURRENT

Miscellaneous

ACTION: Returns the project drive current setting.

PROGRAM SYNTAX: CURRENT cr

REMARKS: This command only applies to the SS2000D6i controlller. It returns the Drive Current setting in amperes of the user project.

EXAMPLES:

```
CURRENT          'Returns the Drive Current setting
```

DECEL

Trajectory Parameters

ACTION: Sets or returns the deceleration value.

PROGRAM SYNTAX:

```
DECEL=number cr
DECEL cr
```

ABBREVIATION: Abbreviation **DC** can be used in place of DECEL.

REMARKS: Setting DECEL less than or equal to zero does not set Error Code 7.

See Programming Command **DECEL.**

EXAMPLES:

```
DECEL=3.1        'sets the deceleration value to 3.1 units/sec2.
```

DIR

ACTION: Returns the user project information.

PROGRAM SYNTAX: DIR cr

REMARKS: If there is no user project, DIR returns a **crlf**.

Returns the following ASCII format:

VER n.nn
pppppppp mm\dd\yyyy hh:mm
where:

n.nn	project compiler version.
pppppppp	project name. Up to 8 characters can be used for a project name. If less than 8 characters is used to identify a project the trailing characters will be spaces.
mm	month the project was compiled.
dd	day the project was compiled.
yyyy	year the project was compiled.
hh	hour the project was compiled.
mm	minutes the project was compiled.

EXAMPLES:

```
DIR cr                  ' with no project loaded
        crlf

DIR cr                  ' with project test1 loaded
        VER 1.00crlf
        test1   06\26\1996 12:30
```

compiled with version 1.00 compiler. Project name is **test1**. compiled June 26 1996 at 12:30.

DIST

Trajectory Parameters

ACTION: Returns the distance moved from the start of the last commanded motion or changes the move distance during indexed motion.

PROGRAM SYNTAX:

```
DIST=number  cr
DIST cr
```

REMARKS: See Programming Command **DIST.**

EXAMPLES:

```
DIST          ' returns the distance traveled from the start of motion.

MOVEI=-25
DIST=-10      ' shorten the move by 10 units
```

ENCPOS

Trajectory Parameters

ACTION: Returns the encoder position.

PROGRAM SYNTAX:

```
ENCPOS  cr
```

ABBREVIATION: Abbreviation **EP** can be used in place of ENCPOS.

REMARKS: See Programming Command **ENCPOS.**

EXAMPLES:

```
ENCPOS          ' returns the encoder position.
```

ENCSPD

Trajectory Parameters

ACTION: Returns the current encoder speed in units/second.

PROGRAM SYNTAX: ENCSPD cr

ABBREVIATION: Abbreviation **ES** can be used in place of ENCSPD.

REMARKS: See Programming Command **ENCSPD.**

EXAMPLES:

```
ENCSPD        ' returns the current encoder speed.
```

ERR

Miscellaneous

ACTION: Returns the error status of the controller.

PROGRAM SYNTAX: ERR cr

REMARKS: See Programming Command **ERR.**

EXAMPLES:

```
ERR cr ' return the error status.
```

ERRM

Miscellaneous

ACTION: Returns the error status of the controller.

PROGRAM SYNTAX: ERRM cr

REMARKS: See Programming Command **ERR.**

EXAMPLES:

```
ERRM cr       ' return the error status.
```

"ESCAPE"

Immediate

ACTION: The ESCAPE command is used during program execution to allow host commands to be executed.

PROGRAM SYNTAX: Press the ESCAPE key or send ASCII code 27.

REMARKS: The ESCAPE command must precede a host command during program execution in order for it to be executed.

Note: This command is placed in quotes. This is because it is a keypress or ASCII code and is not spelled (typed) out in letters. The ASCII code may be sent to the controller if a keyboard is not used.

EXAMPLES:

```
"ESCAPE" ABSPOS cr        ' returns the absolute position
"ESCAPE" STOP cr          ' stops motion
"ESCAPE" FEEDHOLD cr      ' Feedhold motor
```

EVENT1

Motion

ACTION: Sets the trigger polarity and trigger enable, which are used in a MOVEHOME and MOVEREG cycle.

PROGRAM SYNTAX: EVENT1=number cr

ABBREVIATION: Abbreviation **E1** can be used in place of EVENT1.

REMARKS: See Programming Command **EVENT1**

EXAMPLES:

```
EVENT1=0      'disables EVENT1 trigger if assigned as a
              MOVEREG or MOVEHOME trigger.
EVENT1=1      'Sets  EVENT1 trigger to positive edge triggering and
              enables the trigger.
EVENT1=-1     'Sets  EVENT1 trigger to negative edge triggering and en-
              ables the trigger.
```

EVENT2

Motion

ACTION: Sets the trigger polarity and trigger enable, which are used in a MOVEHOME and MOVEREG cycle.

PROGRAM SYNTAX: EVENT2=number cr

ABBREVIATION: Abbreviation **E2** can be used in place of EVENT2.

REMARKS: See Programming Command **EVENT2**

EXAMPLES:

EVENT2=0
Disables EVENT2 trigger if assigned as a MOVEREG trigger.

EVENT2=1
Sets EVENT2 trigger to positive edge triggering and enables the trigger.

EVENT2=-1
Sets EVENT2 trigger to negative edge triggering and enables the trigger.

FEED HOLD

Motion

ACTION: This command stops all motion by decelerating at the programmed decel rate. The user program continues to run. To continue motion issue a RUN host command or toggle the RUN hardware input from inactive to active.

PROGRAM SYNTAX: FEEDHOLD cr

ABBREVIATION: Abbreviation **FH** can be used in place of FEEDHOLD.

REMARKS:

FEEDHOLD
Control stops any motion. The control stopping rate is the programmed DECEL rate.

To resume the stopped motion issue a **RUN** host command.

To cancel the motion issue a **<Ctrl-A>** host command.

EXAMPLES: FEEDHOLD cr

FOLERR

Trajectory Parameters

ACTION: Sets or returns the position error limit. When the position error limit is exceeded action is taken according to the error action parmeter in the closed loop setup and configuration folder.

PROGRAM SYNTAX:
FOLERR=number cr
FOLERR cr

ABBREVIATION: Abbreviation **FE** can be used in place of FOLERR.

REMARKS: Entering a negative value does not set error code 19.

See Programming Command **FOLERR.**

EXAMPLES:
```
FOLERR=.5          ' position error limit is set to .5 units
```

FREEMEM

Miscellaneous

ACTION: Returns the total program space available and the amount of free memory remaining for program storage.

PROGRAM SYNTAX: FREEMEM cr

REMARKS: The return format is:

tttt,nnnn

where: tttt total number of 8 bit bytes available.

nnnn number of 8 bit byte remaining.

An option to save or not save the source code for the project is selected by accessing the **System** menu item **Source Code** in the MX2000-TDC Programming Environment . The saving of the source code results in the compressed source code being added to the compiled project during a project download. If more memory is required to store the project simply select the do not save the source code.

EXAMPLES:
```
FREEMEM cr
      8192,8000
      8192 total bytes available with 8000 bytes remaining.
```

HARDLIM OFF

Travel Limits

ACTION: Disables the hardware limit inputs.

PROGRAM SYNTAX: HARDLIMOFF cr

ABBREVIATION: Abbreviation **HL0** can be used in place of HARDLIMOFF.

REMARKS: See Programming Command **HARDLIMOFF.**

EXAMPLES:

```
HARDLIMOFF          'hard limit inputs are general purpose.
```

HARDLIM ON

Travel Limits

ACTION: Enables the hardware limit inputs.

PROGRAM SYNTAX: HARDLIMON cr

ABBREVIATION: Abbreviation **HL1** can be used in place of HARDLIMON.

REMARKS: See Programming Command **HARDLIMON.**

EXAMPLES:

```
HARDLIMON           'hard limit inputs are active
```

IN

I/O Operator

ACTION: Returns the state of a digital input.

PROGRAM SYNTAX: IN(nn) cr

ABBREVIATION: Abbreviation **I** can be used in place of IN.

REMARKS: Setting **nn** greater than 17 does not set Error Code 9.

See Programming Command **IN.**

EXAMPLES:

```
IN(6)          ' current state of input 6 is returned.
```

JOG

Motion

ACTION: Move in the specified direction.

PROGRAM SYNTAX: JOG = number cr

ABBREVIATION: Abbreviation **J** can be used in place of JOG.

REMARKS: See Programming Command **JOG.**

EXAMPLES:

```
JOG=+1         ' start a jog in the positive direction.
```

LOWSPD

Trajectory Parameter

ACTION: Sets or returns the starting speed value of a stepping motor.

PROGRAM SYNTAX:

LOWSPD=number cr
LOWSPD cr

REMARKS: See Programming Command **LOWSPD**.

EXAMPLES:

```
LOWSPD=2.5          'Sets the starting speed to 2.5 units/sec

LOWSPD              'Returns the current starting speed
```

MOVEA

Motion

ACTION: Initiates the motor to move to the specified absolute position.

PROGRAM SYNTAX: MOVEA = number cr

ABBREVIATION: Abbreviation **MA** can be used in place of MOVEA.

REMARKS: Entering a move distance that exceeds 2,147,483,647 counts does not set Error Code 21.

See Programming Command **MOVEA.**

EXAMPLES:

```
MOVEA=2.5           ' moves to absolute position of 2.5 units.
```

MOVE HOME

Motion

ACTION: Runs the motor until the home input is activated, captures and records the position of the switch activation as home (electrical zero), then stops.

PROGRAM SYNTAX: MOVEHOME = number cr

ABBREVIATION: Abbreviation **MH** can be used in place of MOVEHOME.

REMARKS: See Programming Command **MOVEHOME.**

EXAMPLES:

```
MOVEHOME=+1       'Returns to mechanical home in the Positive direction
```

MOVEI

Motion

ACTION: Initiates an incremental move.

PROGRAM SYNTAX: MOVEI = number cr

ABBREVIATION: Abbreviation **MI** can be used in place of MOVEI.

REMARKS: See Programming Command **MOVEI.**

EXAMPLES:

oves +2.5 units.

MOVEREG

Motion

ACTION: Runs the motor until the mark registration input is activated, then moves the motor the desired registration distance without stopping.

PROGRAM SYNTAX: MOVEREG =number cr

ABBREVIATION: Abbreviation **MR** can be used in place of MOVEREG.

REMARKS: See Programming Command **MOVEREG.**

EXAMPLES:

```
MOVEREG=+2.5        'Initiates a registration cycle in the positive direction with
                    a move of 2.5 units after the mark registration input is activated.
```

OUT

I/O Operator

ACTION: Sets or returns the state of a specified digital output.

PROGRAM SYNTAX:

```
OUT(n) = number cr
OUT(n) cr
```

ABBREVIATION: Abbreviation **O** can be used in place of OUT.

REMARKS: Setting **n** greater than 8 does not set Error Code 8.

See Programming Command **OUT.**

EXAMPLES:

```
OUT(1)=1            ' sets OUT 1 to the active state.

OUT(2)=0            ' sets OUT 2 to the inactive state
```

REGLIMIT

Travel Limits

ACTION: Sets or returns the maximum mark registration distance before indicating an error and stopping motion.

PROGRAM SYNTAX:
REGLIMIT cr
REGLIMIT=number cr

ABBREVIATION: Abbreviation **RL** can be used in place of REGLIMIT.

REMARKS: See Programming Command **REGLIMIT.**

EXAMPLES:

```
REGLIMIT=0          ' disables the MOVEREG travel distance limit.

REGLIMIT=10         ' set the MOVEREG travel distance limit to 10 units.

 REGLIMIT           ' returns the current REGLIMIT value
```

RESET

Miscellaneous

ACTION: Resets the system.

PROGRAM SYNTAX:
RESET cr

REMARKS: This command causes the system to halt, and then restart as though power had been cycled.

REVISION

Miscellaneous

ACTION: Returns the current revision level of the controller's operating system software.

PROGRAM SYNTAX: REVISION cr

ABBREVIATION: Abbreviation **REV** can be used in place of REVISION.

REMARKS: The return format for this command is:
DCC REV n.nn mm/dd/yy

where:

n.nn	software revision
mm	month
dd	day
yy	year

EXAMPLES: REV

RUN

Miscellaneous

ACTION: Starts the user program or resumes from a FEEDHOLD condition which has been generated either by the hardware input **Feedhold** or by the **FEEDHOLD** host command.

PROGRAM SYNTAX: RUN cr

REMARKS: Starts execution of the user program if a Feedhold condition does not exist.

Resumes motion if a Feedhold condition exists.

EXAMPLES: RUN

SOFTLIM NEG

Travel Limits

ACTION: Programmable "software limit switch" for motion in the negative direction. Sets or returns the absolute negative limit travel position value.

PROGRAM SYNTAX:
SOFTLIMNEG=number cr
SOFTLIMNEG cr

ABBREVIATION: Abbreviation **SLN** can be used in place of SOFTLIMNEG.

REMARKS: See Programming Command **SOFTLIMNEG.**

EXAMPLES:

```
SOFTLIMNEG= -4    ' sets the absolute software travel limit to -4 units.

SOFTLIMNEG        ' returns the software travel limit value
```

SOFTLIM OFF

Travel Limits

ACTION: Disables the software over travel limits.

PROGRAM SYNTAX: SOFTLIMOFF cr

ABBREVIATION: Abbreviation **SL0** can be used in place of SOFTLIMOFF.

REMARKS: See Programming Command **SOFTLIMOFF.**

EXAMPLES:

```
SOFTLIMOFF        'Disables the negative and positive software limits
```

SOFTLIM ON

Travel Limits

ACTION: Enables the software over travel limits.

PROGRAM SYNTAX: SOFTLIMON cr

ABBREVIATION: Abbreviation **SL1** can be used in place of SOFTLIMON.

REMARKS: See Programming Command **SOFTLIMON.**

EXAMPLES:

```
SOFTLIMON              'Enables the negative and positive software limits.
```

SOFTLIM POS

Travel Limits

ACTION: Programmable "software limit switch" for motion in the positive direction. Sets or returns the absolute positive limit travel position value for the motor.

PROGRAM SYNTAX: SOFTLIMPOS=number cr
SOFTLIMPOS cr

ABBREVIATION: Abbreviation **SLP** can be used in place of SOFTLIMPOS.

REMARKS: See Programming Command **SOFTLIMPOS.**

EXAMPLES:

```
SOFTLIMPOS=4       ' sets the absolute travel distance to +4 units.

SOFTLIMPOS         ' returns the current SOFTLIMPOS value.
```

SPEED

Trajectory Par Motion

ACTION: Sets and returns the target velocity of the motor.

PROGRAM SYNTAX:
SPEED = number cr
SPEED cr

ABBREVIATION: Abbreviation **SPD** can be used in place of SPEED.

REMARKS: See Programming Command **SPEED.**

EXAMPLES:

```
SPEED=2.0          'sets target velocity to 2 units/sec.

SPEED              ' returns 2.0
```

STOP

Motion

ACTION: Stops any motion with a controlled stop.

PROGRAM SYNTAX: STOP cr

ABBREVIATION: Abbreviation **S** can be used in place of STOP.
See programming command STOP.

REMARKS: Stop the motor using the programmed decel and velocity profile.

EXAMPLES:

```
STOP               ' generates a motion stop command.
                   ' the present value of DECEL is used
                   ' as the deceleration rate
```

STOPERR

Motion

ACTION: Sets or returns the maximum position error allowed when motion is stopped, referred to herein as "position error band".

PROGRAM SYNTAX:
STOPERR=number cr
STOPERR cr

REMARKS: See Programming Command **STOPERR**.

EXAMPLES:

```
STOPERR=.1        'sets the position error band to .1 units.
STOPERR           'returns the current STOPERR.
```

WNDGS

Motion

ACTION: Enables or disables the drive.

PROGRAM SYNTAX: WNDGS=number cr

ABBREVIATION: Abbreviation **WN** can be used in place of WNDGS.

REMARKS: See Programming Command **WNDGS.**

EXAMPLES:

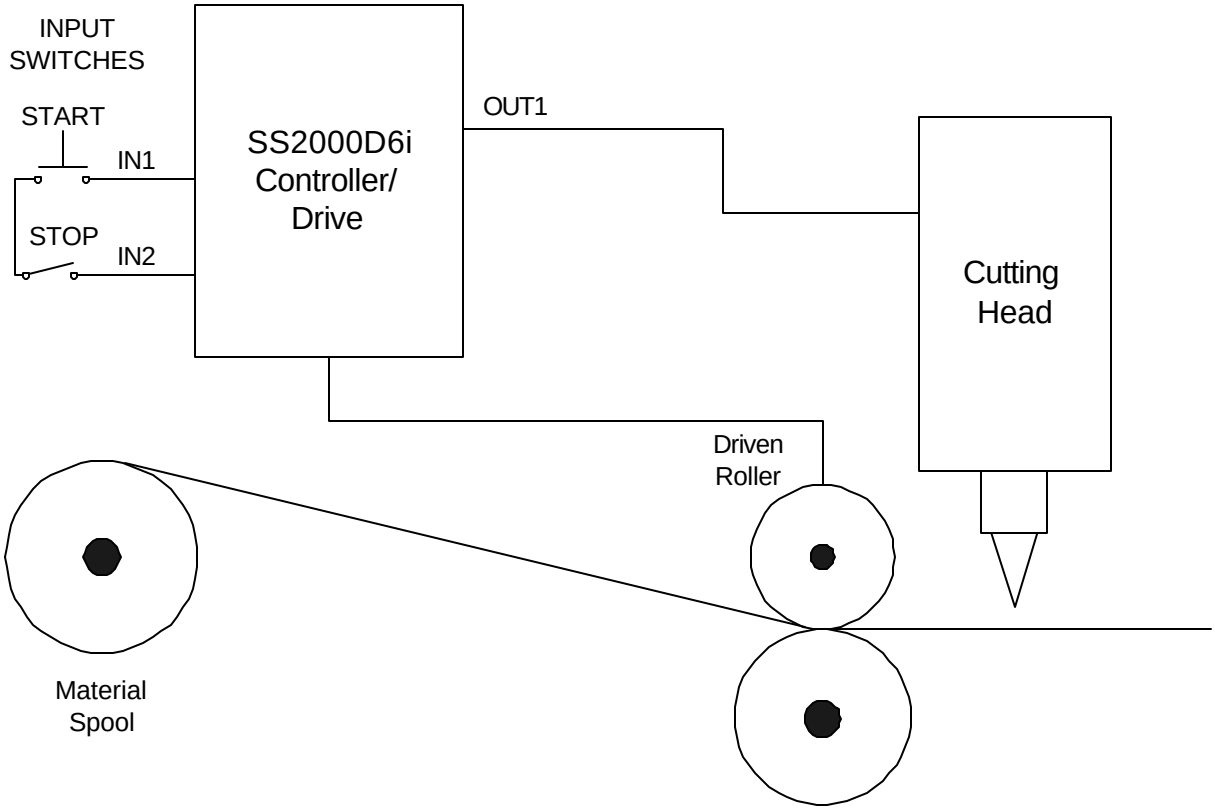
```
WNDGS=0           'disables the drive.
WNDGS=1           'enables the drive.
```

Section 7

Programming Examples

This section contains three sample applications which are easily implemented using the SS2000D6i Controller/Drive. Each sample application contains an explanation of the application including requirements and a diagram. A sample program, including comments, is also provided for each application.

7.1 Cut to Length Application



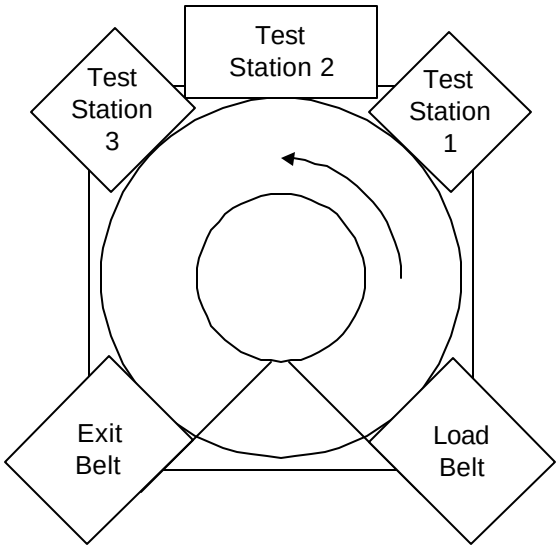
This application requires a stepper motor to run a pair of nip rollers that draw material from a spool. The material could be anything from paper to steel. The requirements for this application are:

- Wait for activation of input 1 (Start switch) from an external device. This device may be an operator input or PLC.
- Feed out a length of material. For this application, a length of 12 inches is required.
- Activate the cutting blade.
- Delay for 1 second. This allows the blade to cut the material.
- Deactivate the cutting blade.
- Delay for 0.25 seconds to allow the blade to return home.
- Repeat the process unless input 2 (Stop switch) is activated.

Program Code:

```
begin:                                  >Label for program return .
        do: loop until (in(1)=1)        >Wait for input 1 to become active.
        movei=12                        >Move 12 inches (incremental move).
        waitdone                        'wait for motion to be completed
        out(1)=1                        >Turn on output 1, cutting blade activation.
        wait=1                          >Wait for 1 second.  Wait for cut to happen.
        out(1)=0                        >Turn off output 1, cutting blade deactivation.
        wait=.25                        >Wait for cutting blade to return, .25 sec.
        if in(2)=0 then goto begin      >Return to beginning of program.
end                                     >End of program.
```

7.2 Rotary Table Application with Test Stations



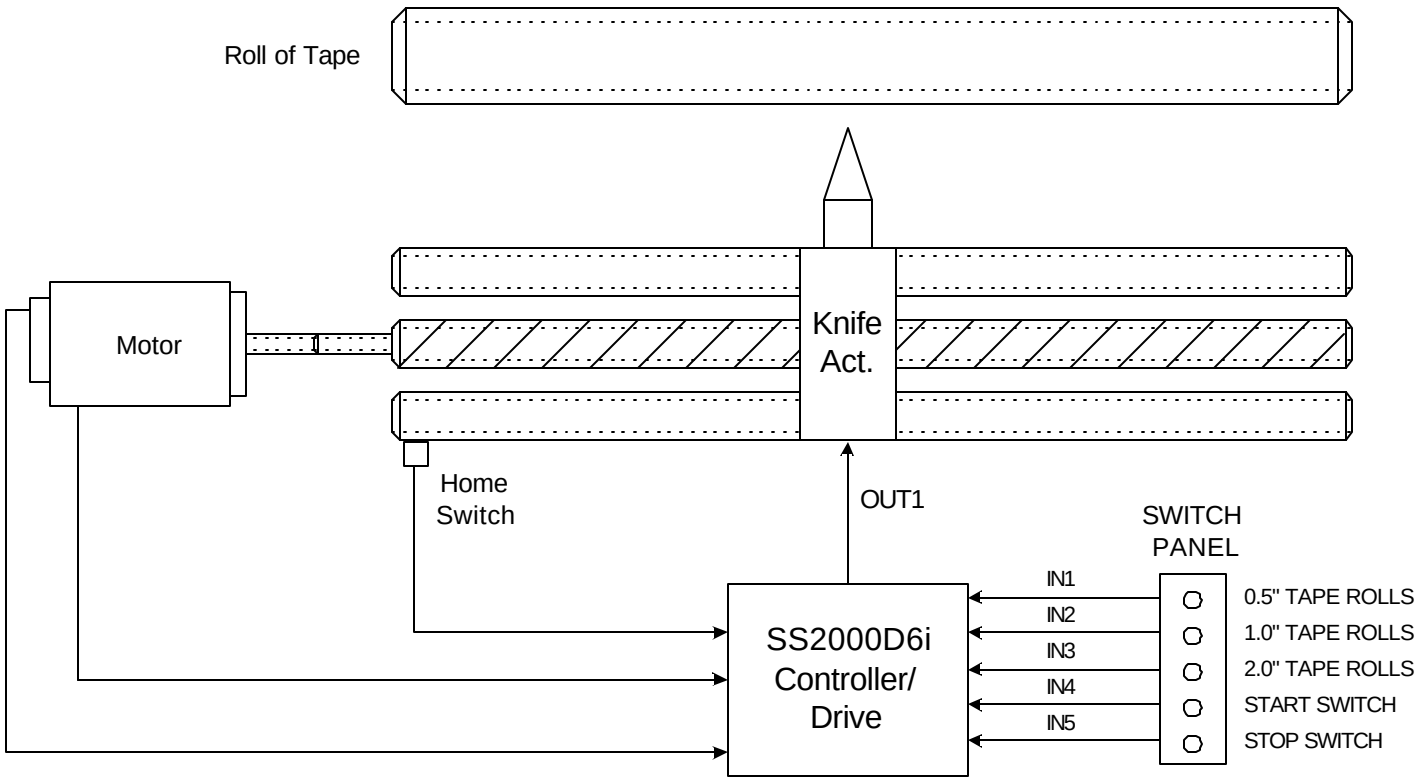
In this application, a part is loaded onto a rotary table via a load belt. Once the part is on the table, it must be tested at three test stations. The application requires:

- A sensor to tell the table to jog until a sensor indicates the presence of a part on the table.
- Turn on outputs to tell the load and exit belts to stop until testing is complete at all the test stations.
- Rotate the table 90E to position the part at the first test station.
- The test procedure requires the sample to be at each station until an input is activated on the control. Each test station is 45E apart.
- After the last test station, the part is rotated 90E to the exit station where it is then carried out via the exit belt. A sensor will then tell the controller to start the load belt for the next part.

Program Code (Rotary Table):

```
integer   count                              > Declare varaiables
begin:                                       > Label to start program
  do:loop until(in(1)=1)                     > Wait until a part is detected by the sensor
    out(2)=1                                 > Turn off the exit belt motor
    out(1)=1                                 > Turn off the loading belt motor
    movei=+90                                > Move table 90E
    waitdone                                 ' Wait for motion to be completed
    do: loop until (in(2)=1)          > Wait for test station to complete testing and turn on input 2.
    count =0                                 > Initialize a counter to zero
    do                                       > Do loop begin
        movei=+45                            > Move rotary table +45E to the next station
        waitdone                             > Wait until the motor stops.
        do: loop  until(in(3)=1 or in(4)=0)  >Stay in this loop until the testing at stations 2 and 3 are complete.
        count=count+1                        > Increment counter by 1.
    loop until count=2                       > Do loop end.
    movei=+90                                > Move the part 90E to the exit belt
    waitdone                                 ' Wait for motion to be completed
    if in(5)=0 then                          > Check input 5 if it is inactive if not continue if active end.
        out(2)=0                             > Turn on exit belt.
        do: loop until in(6)=1               > Stay in loop until the sensor is activated.
        out(1)=0                             > Turn on loading belt.
        goto begin                           > Return to the beginning of the program
    end if                                   > End of the if statement.
end                                          > End of the program.
```

7.3 Slitting Machine Application



A manufacturer of adhesive tape uses a machine that takes a wide roll of tape and slits(cuts) it to the correct sizes. The program must be written to make tape widths of 2 inches, 1inch, and 0.5 inches from a 10 foot long roll. The size of the tape will be determined by the switch inputs from a switch selector panel. The machine will operate as follows:

- Return to mechanical home.
- If input 1 is active make 0.5" wide rolls of tape.
- If input 2 is active make 1.0" wide rolls of tape.
- If input 3 is active make 2.0" wide rolls of tape.
- Loop until a selection is made (switches 1 – 3) and input 4 (start switch) is active
- If input 5 (stop switch) is active end the program else return to electrical home
- Restart program.

Program Code (Slitting machine):

```
integer   I                                   >Declare variables
movehome=1                                    >Return to mechanical home switch.
start:                                        >Start label.
        do: loop until in(4)=1                'Wait for input 4 to be active.
        if in(1)=1 then goto half_in_cut      >If input one is a one goto half_in_cut routine.
        if in(2)=1 then goto one_in_cut       >If input two is a one goto one_in_cut routine.
        if in(3)=1 then goto two_in_cut       >If input three is a one goto two_in_cut routine.
goto start                                    >Return to start.

half_in_cut:                                  >Half inch cut routine.
        for I=1 to 240                        >Beginning of for..next loop.
                movei=.5                      >Move 0.5".
                waitdone                      >Wait until move is completed.
                out(1)=1                      >Turn on the cutting blade on output 1.
                wait=.5                       >Wait for cutter to complete the cut.
        next I                                >End of the for..next loop.
        goto check                            >Goto check subroutine.

one_in_cut:                                   >One inch cut routine.
        for I=1 to 120                        >Beginning of for..next loop.
                movei=1                       >Move 1"
                waitdone                      >Wait until move is completed.
                out(1)=1                      >Turn on the cutting blade on output 1.
                wait=.5                       >Wait for cutter to complete the cut.
        next I                                >End of the for..next loop.
        goto check                            >Goto check subroutine.

two_in_cut:                                   >Two inch cut routine.
        for I= 1 to 60                        >Beginning of for..next loop.
                movei=2                       >Move 2".
                waitdone                      >Wait until move is completed.
                out(1)=1                      >Turn on the cutting blade on output 1.
                wait=.5                       >Wait for cutter to complete the cut.
        next I                                >End of the for..next loop.

check:                                        >Check input 5 subroutine.
        if in(5) = 0 then                     >Check to see if input 5 is inactive.
                movea=0                       >If input 5 is inactive move to electrical home.
                goto start                    >Goto beginning of input search
                end if                        >End of if..then statement
end                                           >End of program.
```

(This Page left intentionally Blank)

Section 8

Troubleshooting Guide

The following table lists some of the more commonly asked troubleshooting questions and possible solutions.

TROUBLESHOOTING QUESTION	POSSIBLE SOLUTIONS
I can=t establish communication with my control. What could be the problem?	1) Insure that the connections are correct. See Section 3.5.3 for serial port communication connections. 2) Check to make sure that the communication parameters are set correctly as defined in Sections 3.1, Switch Settings and 3.4, Serial port 1 switch setting. 3) Check to see if there is power to the unit. Section 3.5.5 provides information on AC power connections.
When I press a button connected to my run input the motor does not turn, why?	1) The switch may not be connected properly. See Section 3.5.2, Input/Output Connections for diagrams on how to properly wire the run input (IN 5). 2) The motor could be connected improperly. Refer to Section 3.5.1, Motor and Encoder Connections, to ensure the motor connectrions are correct. 3) Check to see if there is power to the unit. Section 3.5.5 provides information on AC power connections. 4) The program could also be waiting for an input from another device or switch.
After checking my connections and verifying that they are correct, my motor still does not turn when I start the program.	1) The ENABLE input to the drive may not be properly connected or activated. Insure that the ENABLE input is connected to an active current sinking circuit or switch that is connected with common. See Section 3.4 for information on the User Enable input and Section 3.5.2 for proper connection. 2) Verify that your machine is not mechanically bound-up or obstructed.
When my system activates a sensor the controller does not seem to recognize it, why?	1) Check to make sure that you are operating in the correct mode, i.e. if your using an NPN sensor, make sure you are in sink mode, a PNP sensor requires source mode. Check the connections and make sure you have power to the sensor. Section 3.5.2, Input/Output Connections provides detailed diagrams of proper Sink mode and Source mode wiring connections.

If more information is needed or additional assistance is required contact Superior Electric's, Motion Application Engineering Department at 1-860-585-4510 between 8:00 a.m. and 5:00 p.m. EST

Section 9

Glossary

ABSOLUTE MODE - Motion mode in which all motor movements are specified in reference to an electrical home position.

ABSOLUTE POSITION - A data register in the Controller which keeps track of the commanded motor position. When the value in this register is zero, the position is designated "Electrical Home".

ACCELERATION - The rate at which the motor speed is increased from its present speed to a higher speed (specified in units/second/second).

ACCURACY (of step motor) - The noncumulative incremental error which represents step to step error in one full motor revolution.

AMBIENT TEMPERATURE - The temperature of the air surrounding the motor or drive.

AMPLIFIER - Converts or amplifies low level signals to high voltages and current for use with the motor.

ASCII - (American Standard Code for Information Interchange). A format to represent alphanumeric and control characters as seven-or eight-bit codes for data communications. A table of the ASCII codes appears on the last page of this section.

ATTENTION CHARACTER - <nn, where "nn" is a unique integer from 1-99 (set by use of the unit ID# select switches) that is assigned to a Motion Controller arrayed in a multi-Controller system. The Attention Character directs the program command to the specified Motion Controller.

BACK EMF - The voltage that a permanent magnet motor generates when it is rotated. This has a linear relationship with speed and is related to the voltage constant or back EMF constant of the motor, K_E which is expressed in units of :

$$\frac{\text{Volts}}{\text{1000 RPM}}$$

BANDWIDTH - A given range of frequencies that a motion system can respond to commands.

BAUD RATE - The rate of serial data communications expressed in binary bits per second.

BCD - (Binary Coded Decimal), a format to represent the digits 0 through 9 as four digital signals. Systems using thumb wheel switches may program commands using BCD digits. A BCD digit uses a standard format to represent the digits 0 through 9 as four digital signals.

The following table lists the BCD and complementary BCD representation for those digits. The Motion Controller uses the complementary BCD codes because the signals are active low.

BCD code table (0 = low state, 1=high state)

Digit	BCD Code	Complementary BCD Code
0	0000	1111
1	0001	1110
2	0010	1101
3	0011	1100
4	0100	1011
5	0101	1010
6	0110	1001
7	0111	1000
8	1000	0111
9	1001	0110

To represent numbers greater than 9, cascade the BCD states for each digit. For example, the decimal number 79 is BCD 0111:1001.

BRAKING TORQUE - The torque that is required to bring the motor from a running condition to a stop. This also describes the torque that is developed during a dynamic braking cycle.

CLEAR - Input or Command to immediately halt all motor motion and program execution.

COLLECTORS (OPEN) - A transistor output that takes the signal to a low voltage level with no pull-up device; resistive pull-ups are added to provide the high voltage level.

CYCLE START - Command to initiate program execution.

CYCLE STOP - Command to stop program execution.

DAISY-CHAIN - A method to interface multiple Motion Controllers via RS485 to a single host using only one serial port.

DAMPING - A method of applying additional friction or load to the motor in order to alleviate resonance and ringout.

DECELERATION - The rate in which the motor speed is decreased from its present speed to a lower speed (specified in units/second/second).

DEVICE ADDRESS - A unique number used to assign which Motion Controller in a multi-drive stepper system is to respond to commands sent by a host computer or terminal. Device addresses from 1 - 99 are set by means of the ID # select switch. "00" is reserved to address all Motion Controllers in a system. Factory default is 01.

DUTY CYCLE - The amount of Aon≅ time versus the Aoff≅ time. This is usually expressed in terms of a percentage of the Aon≅ time versus the total time. This is given by the
following equation:

$$D_{Cycle} = \frac{T_{ON}}{T_{OFF} + T_{ON}} \times 100$$

DWELL - See "WAIT".

ELECTRICAL HOME - The location where the motor position counter (abspos) is zero.

ENCODER - a mechanical device attached to the motor that provides a pulse output. This output can be used to determine position, speed or acceleration. The encoder may also be an absolute encoder or incremental.

FEEDBACK - A signal that is transferred from the output, in this case the motor, back to the input where it is compared to see if a particular goal has been achieved.

FEEDHOLD - The act of stopping the motor while in motion by causing it to decelerated to a stop without loss of position.

FEEDRATE - The speed or velocity (in units per second) at which a move will occur.

FRICTION - Force that is opposite to the direction of motion as one body moves over another.

FULL-STEP - Position resolution in which 200 pulses corresponds to one motor revolution in a 200 step per revolution (1.8 degree) motor.

HALF-STEP - Position resolution in which 400 pulses corresponds to one motor revolution for a 200 step per revolution (1.8 degree) motor.

HANDSHAKE - A computer communications technique in which one computer's program links up with another's. The Motion Controller uses a software "Xon, Xoff" handshake method. See "XON" below.

HOST - The computer or terminal that is connected to the HOST serial port on the motion controller, and is responsible for primary programming and operation of the controller.

INCREMENTAL MODE - Motion mode in which all motor movements are specified in reference to the present motor position.

INDEXER - A Microprocessor-based programmable motion controller that controls move distance and speeds; possesses intelligent interfacing and input/output capabilities.

INDEX FROM RUN - See MARK REGISTRATION

INERTIA - Measurement of a property of matter that a body resists a change in speed (must be overcome during acceleration).

INERTIAL LOAD - A "flywheel" type load affixed to the shaft of a step motor. All rotary loads (such as gears or pulleys) have inertia. Sometimes used as a damper to eliminate resonance.

JOG MOVE - moves the motor continuously in a specified direction.

LOAD - This term is used several ways in this and other manuals.

LOAD (ELECTRICAL): The current in Amperes passing through a motor's windings.

LOAD (MECHANICAL): The mass to which motor torque is being applied (the load being moved by the system).

LOAD (PROGRAMMING): Transmits a program from one commuter to another. "DOWNLOAD" refers to transmitting a program from a host computer (where a program has been written) to the Motion Controller where it will be used. "UPLOAD" refers to transmitting a program from a Motion Controller back to the host computer.

MARK REGISTRATION - A motion process (usually used in web handling applications) whereby a mark placed on the material is sensed (e.g., through the use of an optical sensor) and, following detection of this mark, the material is moved (indexed) a fixed length.

MECHANICAL HOME - The position where a switch input is used as a reference to establish electrical home.

MOVE TO MECHANICAL HOME - Function which allows the Motion Controller to move the motor and seek a switch to establish electrical home and set Absolute Position = zero.

NESTING - The ability of an active subroutine to call another subroutine. The Motion Controller can nest up to 16 levels.

NON-VOLATILE MEMORY - Data storage device that retains its contents even if power is removed. Examples are EEPROM, flash memory, and battery-backed RAM.

OPTO-ISOLATION - The electrical separation of the logic section from the input/output section to achieve signal separation and to limit electrical noise. The two systems are coupled together via a transmission of light energy from a sender (LED) to a receiver (photo transistor).

PARITY -- An error checking scheme used in serial communications (via the RS-232 or RS-485 port) to ensure that the data is received by a Motion Controller is the same as the data sent by a host computer or terminal.

REGENERATION - A condition when the motor enters a Abraking≅ mode. The motor acts as a generator because of the transfer of kinetic energy being converted into electrical energy through the motor.

RESOLUTION - The minimum position command that can be executed. Specified in steps per revolution or some equivalent.

RINGOUT - The transient oscillatory response (prior to settling down) of a step motor about its final position. Note: a small wait or dwell time between moves can alleviate ringout problems.

RMS CURRENT - Root Mean Square Current. In an intermittent duty cycle application, the RMS current is equal to the value of steady state current which would produce the equivalent resistive heating over a long period of time.

RMS TORQUE - Root Mean Square Torque. In an intermittent duty cycle application, the RMS torque is equal to the value of steady state torque which would produce the equivalent resistive heating over a long period of time.

RS232-C - EIA (Electronic Industries Association) communication standard to interface devices employing serial data interchanges. Single-wire connections for transmit and receive, etc.

RS-485 - EIA (Electronic Industries Association) communication standard to interface devices employing serial data interchanges. Two-wire connections (differential circuits) for transmit and receive, etc. Better than RS-232 for long wire runs and multi-drop circuits with many devices.

SINKING - An input that responds to, or output that produces, a "low" level (signal common or low side of the input/output power supply) when active.

SOURCING - An input that responds to, or output that produces, a "high" level (the voltage used for the input/output power supply) when active.

SUBROUTINE - A sequence of lines that may be accessed from anywhere in a program to preclude having to program those lines repetitively. This allows shorter, more powerful, and more efficient programs. See also NESTING.

TORQUE - Product of the magnitude of a force and its force arm (radius) to produce rotational movement. Units of measure are pound-inches, ounce-inches, newton-meters, etc.

TORQUE CONSTANT - A number representing the relationship between motor input current and motor output torque. Typically expressed in units of:

$$\frac{\text{torque}}{\text{amps}}$$

TRANSLATOR - A motion control device (also called "translator drive") that converts input pulses to motor phase currents to produce motion.

TTL - Also called T^2L, **T**ransistor - **T**ransistor - **L**ogic

VOLTAGE CONSTANT (or BACK EMF CONSTANT) - A number representing the relationship between the back EMF voltage and angular velocity. Typically expressed in:

$$\frac{\text{Volts}}{\text{1000 RPM}}$$

WAIT - A programmed delay or dwell in program execution (specified in seconds).

XON / XOFF - A computer software "handshaking" scheme used by a Motion Controller.
The Motion Controller sends an XOFF character (ASCII Code 19) when it receives a command string with a Carriage Return and has less than 82 characters remaining in its host serial port buffer. The Controller sends an Xon when available buffer space reaches 100 characters or in response to an ID attention with adequate buffer space remaining. Since it is impossible for the host device to immediately cease transmissions, the next three characters (subject to the total serial buffer capacity of forty characters) received subsequent to the Motion Controller sending the XOFF character will be stored in the Motion Controller's serial buffer (a memory dedicated to store characters that are in the process of transmission).

Similarly, the Motion Controller will not transmit data if the host device has sent an XOFF character to the Controller; Motion Controller transmissions will resume when the Controller receives an XON character.

ASCII Table

ASCII Char	Dec Code	ASCII Char	Dec Code	ASCII Char	Dec Code	ASCII Char	Dec Code
Null	0	Space	32	@	64	`	96
SOH	1	!	33	A	65	a	97
STX	2	A	34	B	66	b	98
ETX	3	#	35	C	67	c	99
EOT	4	$	36	D	68	d	100
ENQ	5	%	37	E	69	e	101
ACK	6	&	38	F	70	f	102
BELL	7	>	39	G	71	g	103
BS	8	(	40	H	72	h	104
HT	9	)	41	I	73	I	105
LF	10	*	42	J	74	j	106
VT	11	+	43	K	75	k	107
FF	12	,	44	L	76	l	108
CR	13	-	45	M	77	m	109
SO	14	.	46	N	78	n	110
SI	15	/	47	O	79	o	111
DLE	16	0	48	P	80	p	112
DC1	17	1	49	Q	81	q	113
DC2	18	2	50	R	82	r	114
DC3	19	3	51	S	83	s	115
DC4	20	4	52	T	84	t	116
NAK	21	5	53	U	85	u	117
SYNC	22	6	54	V	86	v	118
ETB	23	7	55	W	87	w	119
CAN	24	8	56	X	88	x	120
EM	25	9	57	Y	89	y	121
SUB	26	:	58	Z	90	z	122
ESC	27	;	59	[	91	{	123
FS	28	<	60	\	92	|	124
GS	29	=	61	]	93	}	125
RS	30	>	62	^	94	~	126
DEL	31	?	63	_	95	DEL	127

Appendix A
CE Compliance
Installation Requirements and Information

Certain practices must be followed when installing the WARPDRIVE™ SS2000PCi controller in order to meet the CE Electromagnetic Compatibility (EMC) Directive (89/336/EEC) and the Low Voltage Directive (73/23/EEC). The WARPDRIVE™ family of products are components intended for installation within other electrical systems or machines. The system or machine builder must ensure their system or end product complies with all applicable standards required for that equipment, including overall CE certification. Following these practices will help ensure (but cannot guarantee) that the machine in which these components are utilized will meet overall CE requirements.

Electromagnetic Compatibility Directive

In order to meet the various EMC Standards, all wiring must be done in accordance with the practices shown in Figure 1.

With the addition of a suitable ac line filter, such as Corcom part number 10VV1 (connected externally), the SS2000PCi controller meets all the applicable EMC emission and immunity standards on a "stand-alone" basis:

EN55011, Class A:	for Radiated and Conducted Emissions
IEC1000-4-3:	for RF Radiated Immunity (RFRI)
IEC1000-4-4:	for Electrical Fast Transient Immunity (EFT)
IEC1000-4-6:	for RF Conducted Immunity (RFCI)

In order to achieve full CE compliance, an additional requirement must be met:

IEC1000-4-2:	for ESD Immunity

To meet this requirement, the SS2000PCi unit must be placed inside a metal enclosure, as shown in Figure 1.

Low Voltage Directive

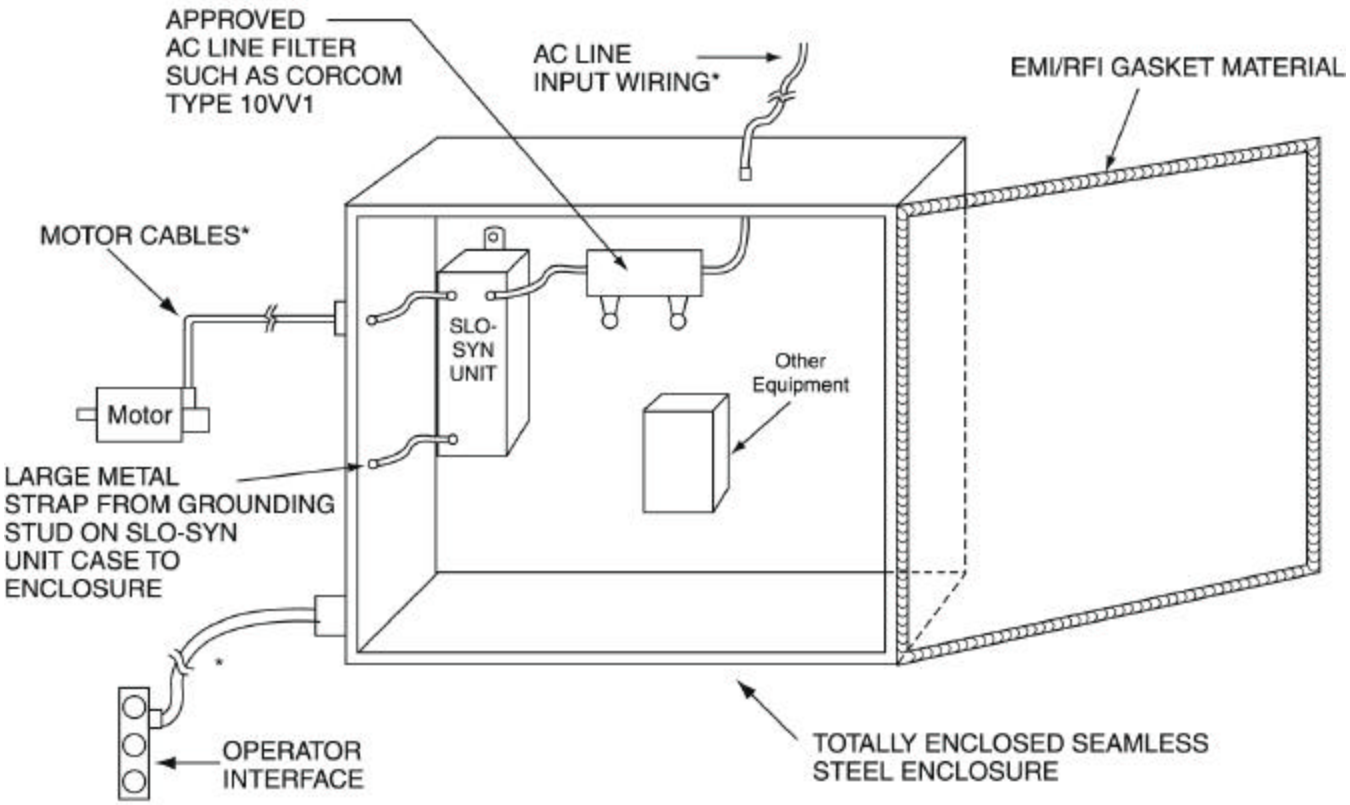
1) These drives are to be operated in a pollution degree 2 environment as described in standard prEN50178.

2) All of the control inputs and outputs are isolated from the main input power with a "basic insulation rating"; e.g., their impulse withstand voltage capability is 2.5kV (1.2 / 50 us) as referenced in prEN50178. Control inputs and outputs may need another level of protection against direct contact if such protection is required by the standards governing the overall system or machine and its intended operating environment. It is the machine-builder's responsibility to provide this protection, if needed.

3) For electrical safety, and to protect personnel against direct contact with live electrical parts, the terminal cover (provided with the unit) MUST be installed over the AC input, motor output, and External REGEN terminals.

4) All cautions and warnings listed throughout the operators manual **MUST** be followed to insure safe system operation.

Figure 1:
Installation for EC EMI/RFI Compliance



* ALL WIRING RUNS THROUGH METAL JACKETED CONDUIT WHICH ARE ATTACHED TO ENCLOSURE WITH CLAMPS MAKING SOUND ELECTRICAL CONTACT.

NOTES:

1) All metal mating surfaces within the enclosure, and any mounting plates should be cleaned of paint, anodizing and coating material for proper electrical bonding. This includes mounting of SLO-SYN unit, line filter, and any other equipment.

2) If mounting plates are used, proper electrical contact to the main enclosure must be maintained. Using copper straps with length-to-width ratios less than 3 is optimum.

WARRANTY AND LIMITATION OF LIABILITY

Superior Electric (the "Company"), Bristol, Connecticut, warrants to the first end user purchaser (the "purchaser") of equipment manufactured by the Company that such equipment, if new, unused and in original unopened cartons at the time of purchase, will be free from defects in material and workmanship under normal use and service for a period of one year from date of shipment from the Company's factory or a warehouse of the Company in the event that the equipment is purchased from the Company or for a period of one year from the date of shipment from the business establishment of an authorized distributor of the Company in the event that the equipment is purchased from an authorized distributor.

THE COMPANY'S OBLIGATION UNDER THIS WARRANTY SHALL BE STRICTLY AND EXCLUSIVELY LIMITED TO REPAIRING OR REPLACING, AT THE FACTORY OR A SERVICE CENTER OF THE COMPANY, ANY SUCH EQUIPMENT OR PARTS THEREOF WHICH AN AUTHORIZED REPRESENTATIVE OF THE COMPANY FINDS TO BE DEFECTIVE IN MATERIAL OR WORKMANSHIP UNDER NORMAL USE AND SERVICE WITHIN SUCH PERIOD OF ONE YEAR. THE COMPANY RESERVES THE RIGHT TO SATISFY SUCH OBLIGATION IN FULL BE REFUNDING THE FULL PURCHASE PRICE OF ANY SUCH DEFECTIVE EQUIPMENT. This warranty does not apply to any equipment which has been tampered with or altered in any way, which has been improperly installed or which has been subject to misuse, neglect or accident.

THE FOREGOING WARRANTY IS IN LIEU OF ANY OTHER WARRANTIES, EXPRESS OR IMPLIED, INCLUDING, WITHOUT LIMITATION, ANY IMPLIED WARRANTY OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE, and of any other obligations or liabilities on the part of the Company; and no person is authorized to assume for the Company any other liability with respect to equipment manufactured by the Company. The Company shall have no liability with respect to equipment not of its manufacture. **THE COMPANY SHALL HAVE NO LIABILITY WHATSOEVER IN ANY EVENT FOR PAYMENT OF ANY INCIDENTAL OR CONSEQUENTIAL DAMAGES, INCLUDING, WITHOUT LIMITATION, DAMAGES FOR INJURY TO ANY PERSON OR PROPERTY.**

Written authorization to return any equipment or parts thereof must be obtained from the Company. The Company shall not be responsible for any transportation charges.

IF FOR ANY REASON ANY OF THE FOREGOING PROVISIONS SHALL BE INEFFECTIVE, THE COMPANY'S LIABILITY FOR DAMAGES ARISING OUT OF ITS MANUFACTURE OR SALE OF EQUIPMENT, OR USE THEREOF, WHETHER SUCH LIABILITY IS BASED ON WARRANTY, CONTRACT, NEGLIGENCE, STRICT LIABILITY IN TORT OR OTHERWISE, SHALL NOT IN ANY EVENT EXCEED THE FULL PURCHASE PRICE OF SUCH EQUIPMENT.

Any action against the Company based upon any liability or obligation arising hereunder or under any law applicable to the sale of equipment, or the use thereof, must be commenced within one year after the cause of such action arises.

Distribution Coast-To-Coast and International

Superior Electric SLO-SYN products are available worldwide through an extensive authorized distributor network. These distributors offer literature, technical assistance and a wide range of models off the shelf for fastest possible delivery and service.

In addition, Superior Electric sales and application engineers are conveniently located to provide prompt attention to customers' needs. Call Superior Electric customer service for ordering and application information or for the address of the closest authorized distributor for Superior Electric's SLO-SYN products.

In U.S.A. and Canada
Superior Electric

- **Customer Service: 1-800-787-3532**
- **Product Application: 1-800-787-3532**
- **Product Literature Request: 1-800-787-3532**
- **Fax: 1-800-766-6366**
- **Web Site: www.danahermotion.com**

383 Middle Street
Bristol, CT 06010
Tel: 860-585-4500
Fax: 860-589-2136

Superior Electric
383 MIDDLE STREET BRISTOL, CT 06010
(860) 585-4500 FAX: (860) 589-2136

ISO 9001 BGB

User Questionaire

Application *(Please fill in the box next to your response.)*

1. *How did you receive this manual?*

☐ included with product ☐ distributor ☐ Superior sales ☐ Other

2. *How frequently do you refer to this manual?*

☐ quarterly ☐ monthly ☐ weekly ☐ daily ☐ seldom

3. *How do you use this manual?*

☐ sit down and read ☐ lookup information ☐ for installation only

☐ to solve problems ☐ other _____

_____

4. *How easy is it to find information in this manual?*

☐ Easy, information is easily retrieved ☐ Somewhat easy, information is there but hard to find

☐ Difficult, (Please explain why)_____

_____

_____

Usability

5. *Please evaluate the following components of this manual, using a 1 to 4 scale. 1 is not descriptive and detailed and 4 is very descriptive and detailed. Circle your selection.*

The table of contents is	(worse)	1	2	3	4 (better)
The headings in the manual are		1	2	3	4
The sub-headings in this manual are		1	2	3	4
The illustrations, graphics in this manual are		1	2	3	4

6. *Please evaluate the overall readability of this manual. 1 is hard to read/understand, 4 is easy to read/understand.*

The table of contents is	1	2	3	4
The headings in this manual are	1	2	3	4
The instructions are	1	2	3	4
The illustrations/graphics are	1	2	3	4

7. *Evaluate the accuracy of this manual. 1 indicates not accurate, 4 indicates accurate.*

The table of contents is	1	2	3	4
The headings in this manual are	1	2	3	4
The instructions are	1	2	3	4
The illustrations/graphics are	1	2	3	4

8. *Which sections, if any, are inaccurate?* _____

_____

_____

9. *Which illustrations/graphics, if any, are inaccurate?* _____

_____

_____

Overall Impression *(please mark all that apply.)*

10. *What do you like/dislike about this manual?*

Like	Dislike		Like	Dislike		Like	Dislike	
☐	☐	Graphics	☐	☐	Illustrations	☐	☐	Length
☐	☐	Format	☐	☐	Special Sections	☐	☐	Size
☐	☐	Express Start Up	☐	☐	A particular section	☐	☐	Detail

11. *Overall Comments:*

What would you change? _____

_____

_____

What sections need to be updated? _____

_____

What is your overall impression of this manual? _____

_____

_____

Please remove this page from the manual, fold, seal with tape and return, thank you.

Name (optional): _____

Company (optional): _____

Phone (optional): _____

Email Address (optional): _____

May we contact you? ☐ Yes ☐ No

fold here

fold here

SS2000PCi

NO POSTAGE NECESSARY IF MAILED IN THE UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS MAIL PERMIT NO. 200 BRISTOL, CT

POSTAGE WILL BE PAID BY ADDRESSEE

SUPERIOR ELECTRIC
383 MIDDLE STREET
BRISTOL, CT 06010-9933