

ARTISAN®
TECHNOLOGY GROUP

Your **definitive** source for quality pre-owned equipment.

Artisan Technology Group
(217) 352-9330 | sales@artisantg.com | artisantg.com

Full-service, independent repair center
with experienced engineers and technicians on staff.

We buy your excess, underutilized, and idle equipment
along with credit for buybacks and trade-ins.

Custom engineering
so your equipment works exactly as you specify.

- Critical and expedited services
- In stock / Ready-to-ship
- Leasing / Rentals / Demos
- ITAR-certified secure asset solutions

Expert team | Trust guarantee | 100% satisfaction

All trademarks, brand names, and brands appearing herein are the property of their respective owners.

Visit our website - *Click HERE*

Superior Electric

www.DanaherMotion.com

SLO-SYN® WARPDRIVE™ SS2000D6i/D3i Programmable Step Motor Controller Installation and Operating Instructions

Document Number: 400030-106
Rev F

DANAHER MOTION

Record of Manual Revisions

ISSUE NO.	DATE	BRIEF DESCRIPTION OF REVISION
A	04/14/98	Initial release
B	07/31/98	Revise section 7 to add variable declarations. Add CE compliance information.
C	02/10/99	Revise to add SS2000D3i where appropriate.
D	02/24/00	Revise corporate identity
E	01/15/03	Revise corporate identity and reformat.
F	01/15/03	Revise corporate identity and reformat.

Copyright Information

© Copyright 1998 - 2003 Danaher Motion - All rights reserved.
Printed in the United States of America.

NOTICE:
Not for use or disclosure outside of Danaher Motion except under written agreement. All rights are reserved. No part of this book shall be reproduced, stored in retrieval form, or transmitted by any means, electronic, mechanical, photocopying, recording, or otherwise without the written permission from the publisher. While every precaution has been taken in the preparation of the book, the publisher assumes no responsibility for errors or omissions. Neither is any liability assumed for damages resulting from the use of the information contained herein.

This document is proprietary information of Danaher Motion that is furnished for customer use ONLY. No other uses are authorized without written permission of Danaher Motion. Information in this document is subject to change without notice and does not represent a commitment on the part of Danaher Motion. Therefore, information contained in this manual may be updated from time-to-time due to product improvements, etc., and may not conform in every respect to former issues.

SAFETY

Before installing and operating your SLO-SYN motion control product, it is extremely important that you read and understand this safety section. The SLO-SYN product will deliver years of reliable, trouble-free, safe operation if you heed the cautions and warnings as well as follow the instructions contained in this manual.

Safe symbols used in this manual are:



Warnings are provided to alert you to possible damage to equipment and potential electrical shock hazards to personnel.



Cautions are provided to alert you to possible damage to equipment.



Notes alert you to helpful information.

GENERAL SAFETY INFORMATION



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! Exercise caution when installing and using this unit. Only qualified personnel should attempt to install and/or operate this product. It is essential that proper electrical practices, applicable electrical codes, and the information in this manual be strictly followed.



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



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



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



The heatsink or unit could be hot to the touch. Use caution when determining the temperature.



Secure mounting and proper grounding of the SLO-SYN unit 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 when using this product.



Do not use ANY of the unit's inputs as an EMERGENCY STOP in ANY application. Although activation of certain inputs 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 (IEC 204-1). Other measures (mechanical stops, fail-safe brakes) must be used.



Surface temperatures of step motors can reach or exceed 100 °C (212 °F). Use caution when handling motors.



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 or 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.

Customer Support

Danaher Motion products are available nationwide 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.

Danaher Motion sales engineers are conveniently located to provide prompt attention to customers' needs. Call the nearest office listed for ordering and application information or for the address of the closest authorized distributor.

In the US and Canada
13500-J South Point Blvd.
Charlotte, NC 28273
Phone: (704) 588-5693
Fax: (704) 588-5695
Email: sales2@danahermotion.com
Website: www.DanaherMotion.com

In Europe
Danaher Motion GmbH & Co. KG
Robert-Bosch-Straße 10
64331 Weiterstadt, Germany
Phone: +49 (0) 6151-8796-10
Fax: +49 (0) 6151-8796-123
Website: www.DanaherMotion.de



Manufactured Under an ISO9001
Compliant Manufacturing System

Contents

1. QUICK START INSTALLATION 1

1.1. BENCH SETUP 1
1.2. INSTALLATION INTO MECHANICAL SYSTEM 4
1.2.1. SWITCH SETTINGS 4
1.2.2. BAUD RATE AND UNIT ID SWITCH 4
1.2.3. SS2000D3I OUTLINE DIMENSIONS 6
1.2.4. SS2000D6I OUTLINE DIMENSIONS 7
1.3. MECHANICAL MOUNTING 7
1.4. GENERAL WIRING GUIDELINES 8
1.5. WIRING GUIDELINES FOR CE COMPLIANCE 9
1.6. HARDWARE CONNECTIONS 11
1.7. WIRING DIAGRAMS 19
1.7.1. MOTOR AND ENCODER CONNECTIONS 20
1.7.2. MOTOR WIRING CONFIGURATION 21
1.7.3. INPUT/OUTPUT CONNECTIONS 22
1.7.3.1. INTERNAL POWER SUPPLY 22
1.7.3.1.1. Inputs Sinking, Outputs Sinking 22
1.7.3.1.2. Inputs Sourcing, Outputs Sinking 23
1.7.3.2. EXTERNAL POWER SUPPLY 23
1.7.3.2.1. Inputs Sinking, Outputs Sinking 24
1.7.3.2.2. Inputs Sourcing, Outputs Sinking 25
1.7.4. OPTIONAL EXTERNAL WIRING CARD 26
1.7.4.1. USING BCD PORT AS NON-ISOLATED I/O 28
1.7.5. RS-232 / RS-485 HOST SERIAL COMMUNICATIONS CONNECTIONS 29
1.7.5.1. RS-485 HOST DAISY CHAINING CONNECTIONS 30
1.7.6. RS-485 AUXILIARY COOMMUNICATION CONNECTIONS 31
1.7.7. AC POWER CONNECTIONS TO THE UNIT 32

2. SPECIFICATIONS 33

2.1. MECHANICAL 33
2.2. ENVIRONMENTAL 33
2.3. ELECTRICAL 33
2.3.1. ISOLATED DIGITAL I/O 34
2.3.2. NON-ISOLATED I/O (FOR BCD INTERFACE) 34
2.3.3. SERIAL COMMUNICATION 35
2.3.4. ENCODER CONNECTIONS 35
2.3.5. ANALOG INPUT 36

2.4. HARDWARE EQUIVALENT CIRCUITS 36
2.5. MOTOR COMPATABILTY 38
2.6. MOTOR PERFORMANCE 39
2.7. TYPICAL TORQUE VERSUS SPEED CURVES 40

3. MOTION CONTROLLER PROGRAMMING INTERFACE 49

3.1. PROGRAMMING THE CONTROLLER 49
3.2. HOST COMMANDS 49
3.3. MEMORY 50
3.4. PROJECTS 50
3.4.1. INITIALIZATION 51
3.4.2. MAIN PROGRAM 52
3.4.3. INTERRUPT ROUTINES 52
3.4.4. SUBROUTINE 53
3.4.5. ERROR HANDLER 53
3.5. MOTION CONTROLLER PROGRAMMING INTERFACE (MCPI) 53
3.5.1. MINIMUM COMPUTER REQUIREMENTS 54
3.5.2. INSTALLATION INSTRUCTIONS (WINDOWS 3.1X) 54
3.5.3. INSTALLATION INSTRUCTIONS (WINDOWS 95) 54
3.5.4. STARTING THE PROGRAMMING ENVIRONMENT 54
3.5.4.1. OPENING SCREEN 55
3.5.4.2. SETTING COMMUNICATION PARAMETERS 55
3.5.5. CREATING A NEW PROJECT 56
3.5.6. THE TASK EDITOR 57
3.5.6.1. TASK MENU SCREEN 57
3.5.6.2. TASK EDITOR SCREEN 58
3.5.6.3. EDIT MENU 58
3.5.6.4. SYSTEM MENU 60
3.5.6.5. TERMINAL EMULATION 61
3.5.7. CONFIGURATION AND SETUP FOLDERS 63
3.5.7.1. SYSTEM FOLDER 63
3.5.7.2. PROFILE FOLDER 65
3.5.7.3. ENCODER FOLDER 66
3.5.7.4. OPEN LOOP STEPPER FOLDER 66
3.5.7.5. CLOSED LOOP STEPPER FOLDER 67
3.5.7.6. MECHANICAL HOME & MARK REGISTRATION FOLDER 69
3.5.7.7. I/O FOLDER 69
3.5.8. PREPARING USER PROJECT FOR EXECUTION 71
3.5.8.1. PROJECT SOURCE CODE 71
3.5.8.2. SETTING PROJECT DEBUGGING 71
3.5.8.3. COMPILING A PROJECT 72

3.5.8.4. SELECTING THE CONTROLLER UNIT ID 72
3.5.8.5. DOWNLOADING A PROJECT 72
3.5.8.6. UPLOADING SOURCE CODE 72
3.5.9. DOWNLOADING AN OPERATING SYSTEM 73
3.5.10. OTHER MENUS 73
3.5.10.1. PROJECT MENU 74
3.5.10.2. UTILITY MENU 75
3.5.10.3. WINDOW MENU 75
3.5.10.4. HELP MENU 75
3.5.11. PROJECT COMMAND BUTTONS 76
3.5.12. DEBUG ENVIRONMENT 76
3.5.12.1. DEBUG PROGRAM EXECUTION 77
3.5.12.2. BREAKPOINT SETTING/CLEARING 77
3.5.12.3. VARIABLE WATCH 77
3.5.12.4. TERMINAL WINDOW 78
3.5.12.5. EXIT DEBUG ENVIRONMENT 78

4. SOFTWARE REFERENCE 79

4.1. PROGRAMMING COMMANDS BY FUNCTION 79
4.1.1. ARITHMETIC OPERATORS 79
4.1.2. BOOLEAN OPERATORS 79
4.1.3. INTERRUPT 79
4.1.4. I/O 79
4.1.5. MISCELLANEOUS 79
4.1.6. MOTION 80
4.1.7. OVER TRAVEL LIMIT 80
4.1.8. PROGRAM FLOW CONTROL 80
4.1.9. RELATIONAL OPERATORS 81
4.1.10. STRING MANIPULATION 81
4.1.11. TIME FUNCTIONS 81
4.1.12. TRAJECTORY PARAMETERS 82
4.1.13. VARIABLE DEFINITIONS 82
4.2. HOST COMMANDS BY FUNCTION 83
4.2.1. MOTION 83
4.2.2. TRAJECTORY PARAMETERS 83
4.2.3. I/O 84
4.2.4. TRAVEL LIMITS 84
4.2.5. MISCELLANEOUS 84
4.2.6. IMMEDIATE 84
4.2.7. DAISY CHAINING 85
4.3. MC-BASIC CONVENTIONS 85

4.3.1. ARITHMETIC OPERATORS 85
4.3.2. LOGICAL OPERATORS 85
4.3.3. RELATIONAL OPERATORS 86
4.3.4. BASIC DATA TYPES 86
4.3.4.1. RULES FOR INTEGER DIVISION 87
4.3.4.2. CASE SENSITIVITY 87
4.3.4.3. CALCULATIONS USING TRAJECTORY PARAMETERS AND VARIABLES 88
4.3.4.4. PROGRAM COMMENTS 88
4.4. PROGRAMMING COMMANDS 88
"BACKSPACE" *..... 88*
"CTRL-A" *..... 89*
"CTRL-C" *..... 89*
"ESCAPE" *..... 89*
<NN *..... 90*
ABSPOS *..... 91*
ACCEL *..... 92*
ANALOG *..... 92*
AND *..... 93*
ASC *..... 93*
BCD *..... 94*
BOOST *..... 94*
BUSY *..... 95*
CHR$ *..... 95*
CURRENT *..... 95*
DECEL *..... 96*
DEFINE *..... 97*
DIR *..... 98*
DIST *..... 99*
DO…LOOP *..... 100*
ENCPOS *..... 100*
ENCSPD *..... 101*
END *..... 101*
ERR *..... 102*
ERRM *..... 103*
EVENT1 *..... 104*
EVENT2 *..... 105*
FEEDHOLD *..... 106*
FOLERR *..... 106*
FOR NEXT *..... 107*
FREEMEM *..... 108*

GETCHAR *108*
GOSUB…RETURN *109*
GOTO *109*
HARDLIMOFF *110*
HARDLIMON *110*
HEX$ *111*
HVAL *111*
IF...THEN...ELSE...ENDIF *112*
IN *113*
INCHAR *114*
INCLUDE *114*
INPUT *115*
INSTR *115*
INTROFFN *116*
INTRONN *116*
JOG *117*
LCASE$ *117*
LEFT$ *117*
LEN *118*
LOWSPD *118*
MID$ *118*
MOVEA *119*
MOVE HOME *119*
MOVEI *120*
MOVEREG *120*
NOT *121*
ON...INTRN *122*
OR *125*
OUT *125*
PRINT *126*
PRINT USING *127*
REDUCE *130*
REGLIMIT *131*
RESET *131*
REVISION *132*
RIGHT$ *132*
RUN *132*
SOFTLIMNEG *133*
SOFTLIMOFF *134*
SOFTLIMON *134*

SOFTLIMPOS *135*
SPEED *136*
STOP *136*
STOPERR *137*
STR$ *137*
STRING$ *138*
TIMER *138*
UCASE$ *138*
UNITID *139*
VAL *139*
WAIT *139*
WAITDONE *140*
WNDGS *140*

5. PROGRAMMING EXAMPLES 141

5.1. CUT TO LENGTH APPLICATION 141
5.2. ROTARY TABLE APPLICATION WITH TEST STATIONS 142
5.3. SLITTING MACHINE APPLICATION 143

6. TROUBLESHOOTING 145

7. GLOSSSARY 147

APPENDIX A 157

A.1 CE COMPLIANCE INSTALLATION REQUIREMENTS AND INFORMATION 157
A.1.1 ELECTROMAGNETIC COMPATIBILITY DIRECTIVE 157
A.1.2 LOW VOLTAGE DIRECTIVE 159
A.2 WARRANTY 159

1. QUICK START INSTALLATION

The SS2000D6i/D3i step motor positioning system is a sophisticated and versatile product. Setting up the system can be simple and straight forward if the proper steps are followed. Please use the steps below to set up your unit.

1.1. Bench Setup

Before connecting your SS2000D6i or SS2000D3i and motor to your mechanical system or machine, we recommend that you "bench test" the system. This allows 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 that cause unexpected motion. The bench setup can be used to perform simple motions with an unloaded motor. To perform a bench test:

a) **Wire it up**. Familiarize yourself with the Wiring Diagrams, connecting the AC power, connecting the I/O and other required signals. **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 the wiring of your system. Wire the AC line input, motor and HOST communications to the unit and perform simple motion.



Do not forget to wire the USER ENABLE signal to GND through a switch so you can turn the drive on and off as necessary.

b) **Load software.** You will need to use a PC to program the unit according to your requirements. First, load the MCPI software onto the PC from the floppy disks provided with your unit (run SETUP.EXE on disk 1). Once the software is loaded, run it by double-clicking the MCPI icon.

c) **Create your project**. Create a new project. The project must contain **Configuration** information for your particular system. The program **Task** holds the user program written in a BASIC-like language. Read the *Motion Control Programming Interface* section of this manual. Then, step through the Configuration folders and enter the appropriate data for your system. Save the configuration when you are finished.



For this exercise, the original default settings should work fine

Set the serial port for your PC to the correct port number and baud rate. Set the **Motor Current** parameter at or *below* the nameplate rating on your motor.



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 means that move distances are in motor revolutions (e.g., movej=1 moves one revolution), speed is in revolutions per second and acceleration is in revolutions per second squared. This can later 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 in this manual for other examples. ALL move distances, speeds, accelerations, decelerations, and encoder information are provided in USER UNITS. Be sure you thoroughly understand USER UNITS before continuing.

d) **Compile and download** the project into the unit using the command buttons of the MCPI. 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 terminal mode using the appropriate command button on your 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, enter the speed at which you wish the motor to turn (e.g., 1 revolution per second). To do this, type: `speed=1 <CR>`



<CR> is the Return or Enter key.

Enter the acceleration rate (e.g., 50 revolutions per second squared) by typing: `accel=50<CR>`

Enter the deceleration rate to match the acceleration rate 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 (particularly the **User Enable** input. Verify your configuration settings. Check the motor direction to ensure it meets your requirements. The motor direction can be reversed in the System folder.

f) **Write a BASIC program**. Once you have made a simple move, you are ready to write your task in the MCPI BASIC-like language. Refer to the *Software Reference Guide* section of this manual for a complete description of all the program **Commands**. Start by opening your Task and entering the commands. First, enter the speed, acceleration, deceleration, and movei commands as you did in the previous step. Two additional commands are required to tell the unit that the program is complete after it performs a move. Type:

```
waitdone=10<CR>
End<CR>
```

These two lines are the last lines of the program. Since your program has changed, you must compile and download it into the unit for the changes to take effect. If you receive compilation errors, check your spelling and syntax information using the *Software Reference Guide* section of this manual.

g) **Execute the program**. From the Terminal screen, click 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.

Example

Change the first line of your program, by typing:

```
REAL x <CR>
```

This declares x as a REAL variable. (See the *Motion Control Programming Interface* and *Software Reference Guide* sections of this manual for additional information.)

On the next line, type:

```
x=10 <CR>
```

This assigns the REAL variable x a value to 10.

Change:

```
movei=10
```

to:

```
movei=x
```

Now, the motor moves whatever distance has been assigned to x. Recompile and download your program. Run it. It should operate the same as before, but now, the program is using x as the move distance instead of 10 (as before). Change the value of x to different distance values to verify that it works correctly.

h) **Expand and debug the program**. Now that you have written a simple program, you can add more complexity by adding more commands. You can perform complex looping, access I/O, and motion functions as required.

It is helpful now to use the DEBUG feature of the MCPI. Refer to the *Motion Control Programming Interface* section for a detailed description of the debug mode. If the program is compiled 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, re-compile in Release Mode to speed up program execution.

1.2. Installation Into Mechanical System

Once you have tested everything in a controlled environment, complete the installation to your system. This requires making all the necessary wiring connections for limit switches, additional I/O, analog inputs, encoder, etc. **Start Simple!** Just as you began with a simple move on the bench, start simple and slowly add complexity as you debug your code and gain more confidence in programming. 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.

1.2.1. Switch Settings



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



Before mounting and wiring your SLO-SYN Positioning system, the switches that govern various operating features should be checked or set to their proper positions for your application.

1.2.2. Baud Rate and Unit ID Switch

The Baud Rate switch is accessible through the top of the unit on the left side 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 download project. If the switch is in the 9600 position, the baud rate is forced to 9600 baud **regardless of the project configuration**.

It is possible to communicate to multiple units over the same RS-485 transmission lines. To accomplish this, the D6i/D3i 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. Be sure the power is OFF when changing the switch position.

To change the HOST port communications, slide the RS-232/RS-485 selector switch to the appropriate location. The switch is accessible through an access hold in the top of the unit near the BCD I/O port. If the BCD port is not in use, remove the cut out section of the top label to gain access to the switch. All units must be set to the same baud rate. Further wiring details are included in the Wiring Diagrams.



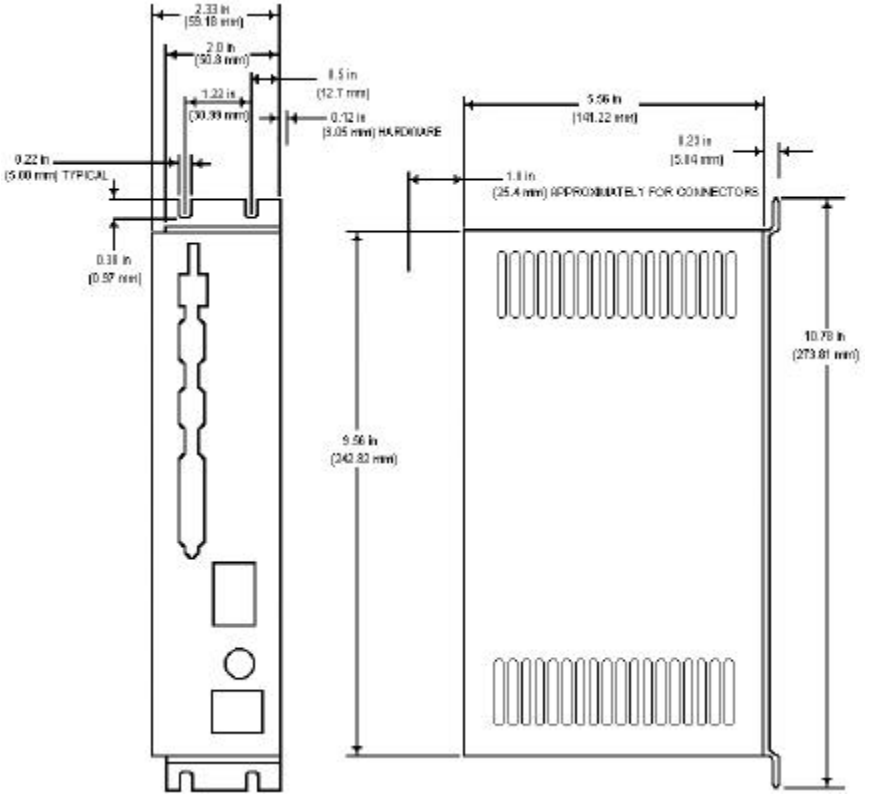
RS-232 daisy-chaining is NOT supported. Do not connect RS-232 signals to the HOST port when it is in RS-485 Mode.

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.

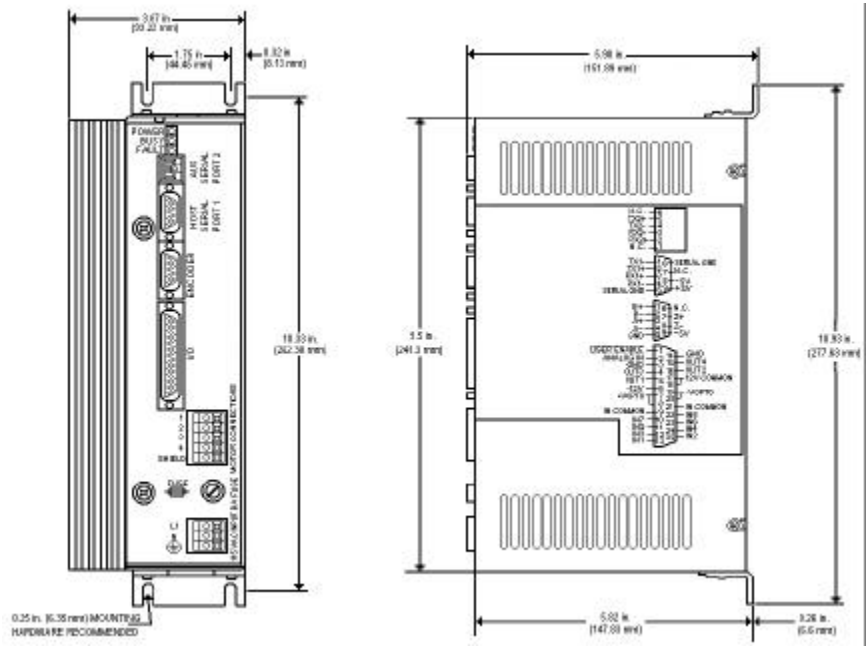
Each unit on the daisy-chain must have a unique identification number (ID) to eliminate transmitter conflicts on the RS-485 port. Five dip switches are provided for selecting the unit ID (1-32). They are accessible through the top rear of the unit. One and ONLY ONE unit MUST have ID 1. The switch positions are only decoded at power up. DO NOT change the switches with the power on! Unit IDs are decoded as:

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

1.2.3. SS2000D3i Outline Dimensions



1.2.4. SS2000D6i Outline Dimensions



1.3. Mechanical Mounting

The Mechanical Outline Drawing provides overall and mounting dimensions for the SS2000D6i and SS2000D3i. The unit should be solidly mounted within a control enclosure approved for the particular application. It is important to select a mounting location that meets the specifications listed in the *Environmental Specifications*. Avoid locations that expose the unit to external temperature, humidity, dirt, dust, or vibration.

At least 2 inches (50.8 mm) of space must be left on the sides, top and bottom of the unit to allow proper airflow for cooling the unit. You must also allow safe access to all wiring. It is best to avoid areas with high electrical noise.

1.4. General Wiring Guidelines



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!

Exercise caution when installing and using this product. Only qualified personnel should attempt to install and operate this product. It is essential that proper electrical practices, applicable electrical codes, and the instructions in this manual be strictly followed.

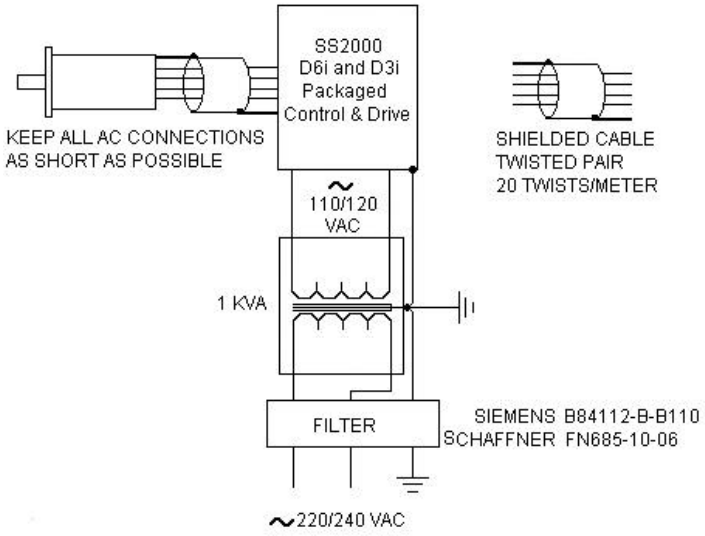
Danaher Motion Superior Electric SLO-SYN controls and drives use modern solid-state digital electronics to provide the features needed for advanced motion control applications. Although care has been taken to ensure proper operation under a wide range of conditions, some user equipment may produce considerable 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
- DC servo and stepper motors and drives

1.5. Wiring Guidelines for CE Compliance

Mandatory connections to meet CE EMC requirements are:



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

- **Solidly ground the system.** Be sure that there is a solid connection to the AC system protective earth ground (PE). Be sure there is a good electrical connection through the drive 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 ½inch (12.7 mm) 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 wire separated.** Power wiring includes AC wiring, motor wires, etc. Signal wiring includes inputs and outputs (I/O), encoder wiring, serial communications (RS-232 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.

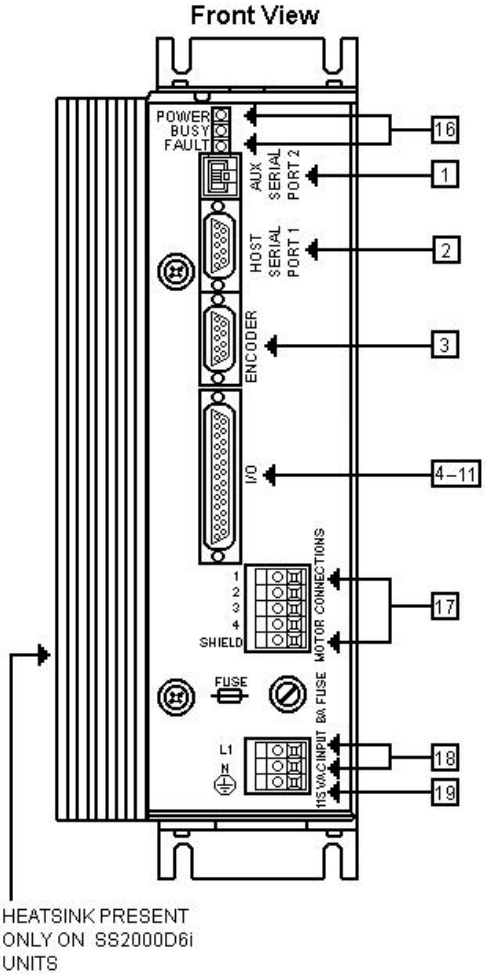
- **Use separately bundled, shielded, twisted-pair cables**. Use these cables for the drive to motor, encoder, serial communications, analog input, and digital I/O wiring. For motor connections, BE SURE TO GROUND THE SHIELD AT THE SLO-SYN DRIVE END. For other connections, it is recommended that the shields 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 ONLY ONE END. 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 MOVs (see manufacturer's literature for complete information). Whenever possible, use solid-state relays instead of mechanical contact types to minimize noise generation.
- **Use external filtering.** In some extreme cases of interference, if may be necessary to add external filtering to the AC line(s) feeding affected equipment or to use isolation transformers to supply the AC power.

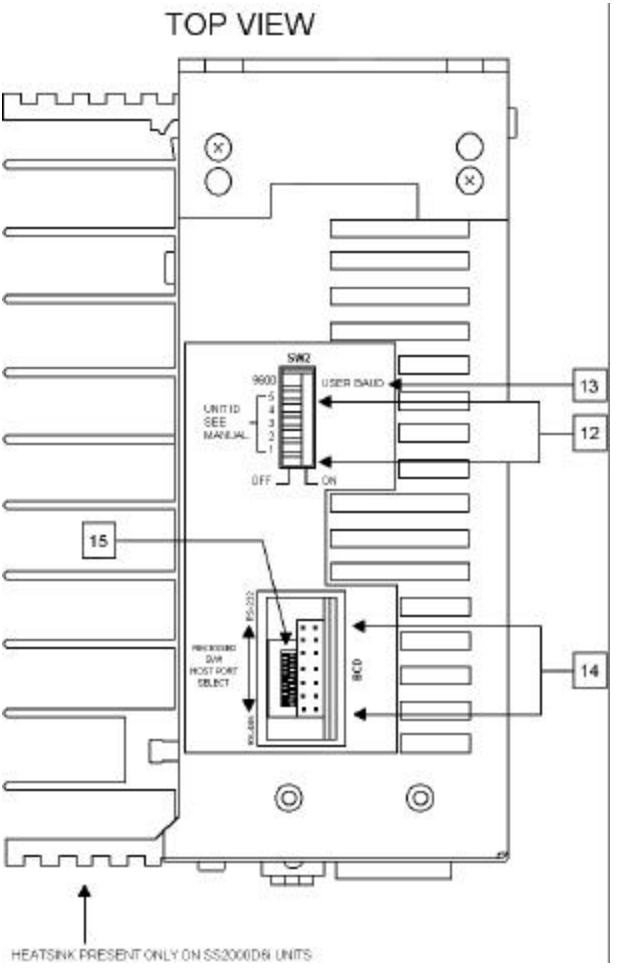


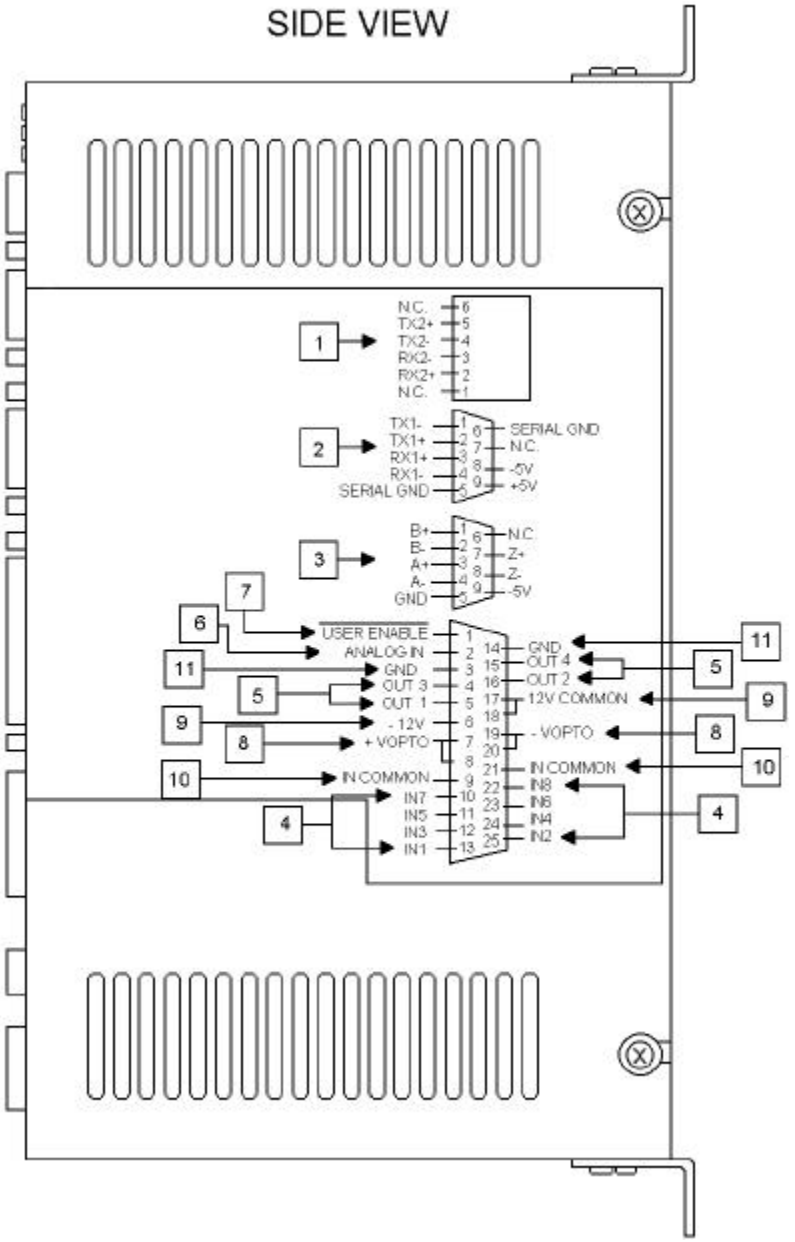
Danaher Motion makes a wide range of AC power line conditioners that can help solve electrical interference problems. Contact Danaher Motion customer support for further assistance.

1.6. Hardware Connections

The following figures indicate the side, top, and front views of the D6i/D3i controller/drive. 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 the *Hardware Specifications* section.




TOP VIEW
SW2
9600
USER BAUD
UNIT ID
SEE
MANUAL
OFF
ON
13
12
15
14
RS-232
BCD
HEATSINK PRESENT ONLY ON SS2000D6 UNITS


SIDE VIEW
1
N.C.
TX2+
TX2-
RX2-
RX2+
N.C.
2
TX1-
TX1+
RX1+
RX1-
SERIAL GND
SERIAL GND
N.C.
-5V
+5V
3
B+
B-
A+
A-
GND
N.C.
Z+
Z-
-5V
7
6
11
5
9
8
10
4
USER ENABLE
ANALOG IN
GND
OUT 3
OUT 1
- 12V
+ VOPTO
IN COMMON
IN7
IN5
IN3
IN1
GND
OUT 4
OUT 2
12V COMMON
- VOPTO
IN COMMON
IN8
IN6
IN4
IN2

The following are brief descriptions of each connection to the D6i and D3i unit. More details are provided in the Wiring Diagrams.

1 Serial Port 2

Auxiliary Serial Communications: Port 2

TX2+:	Transmit + for Serial Port 2 (RS-485)
TX2-:	Transmit - for Serial Port 2 (RS-485)
RX2+:	Receive + for Serial Port 2 (RS-485)
RX2-:	Receive - for Serial Port 2 (RS-485)

2 Serial Port 1

Host Serial Communications: Port 1

TX1+:	Transmit + for Serial Port 1 (RS-485)
TX1-:	Transmit - for Serial Port 1 (RS-485)
RX1+:	Receive + for Serial Port 1 (RS-485)
RX1-:	Receive - for Serial Port 1 (RS-485)
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 Channel Z+ input
Z1-:	Encoder Channel Z- input
+5V:	+5V supply for encoder
GND:	Ground for encoder

4 Inputs 1-8 (Isolated)

Event 1 / IN1; Event 2 / IN2

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

+ LIMIT / IN3; - LIMIT / IN4

The +LIMIT or –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 & Setup* as programmable inputs.

RUN / IN5

The RUN input starts execution of the program. If Autostart is selected, the program starts upon power-up or RESET. RUN also restarts a program if a **CLEAR** has been activated. It resumes a program if a **FEEDHOLD** is activated. If the RUN input is not needed, the input can be configured in the *Project Configuration & Setup* as a programmable input.

CLEAR / IN6

If the **CLEAR** input is open, the program or motion stops. This input MUST be closed to run the program or start motion. If the CLEAR input is not needed, the input can be configured in the *Project Configuration & Setup* as a programmable input.

FEEDHOLD / IN7

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

IN8

This input can be used as a programmable input.

5 Outputs 1-4 (Isolated)

OUT 1, OUT 2, OUT 3, OUT 4

These outputs can be used as programmable outputs.

6 Analog Input

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

ANALOG IN: Analog input

GND: Ground for analog input

7 User Enable

When tied to GND, this hardware signal enables the Stepper Drive. The controller can enable the drive through software. If this input is not connected to GND, no holding torque on the motor or motion is allowed. By default, the controller is in the enabled mode.

8 OPTO

+VOPTO; -VOPTO



A power supply for optical isolators is REQUIRED for proper I/O operation. This supply MUST be connected to the +VOPTO and –VOPTO pins.

The +12 VDC and +12 V 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.

9 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.

+12 V: +12 VDC output

12 V COM: Common for the 12 VDC supply

The +12 VDC supply current is limited to 100 mA. See *Input/Output Connections* for sinking or sourcing I/O.

10 IN COMMON

This input determines the current source of Inputs 1 through 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.

11 GND

Signal common for the **User Enable** and **Analog In** signals.



This GND connection is NOT connected to 12 V COM or IN COMMON.

12 Device ID Number Switch

The DIP switches 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 the *Baud Rate and Unit ID Switches* section.

13 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 *Configuration and Setup*. If no user program is loaded, the default baud rate of 9600 is used. If the baud rate in the configuration and setup is not known, use 9600 at power-up.

14 BCD Port

BCD Port I/O

This port can be used as either a BCE port (consisting of 7 numbers and a sign), or used for additional inputs and outputs.



Danaher Motion Superior Electric Part #221157-002, includes BCD switch and 18-inch (457.2 mm) ribbon cable.

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



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

15 Serial Port 1 (RS-232/RS-485) Switch

This switch allows Serial Port 1 to be configured for RS-232 or RS-485 4-wire communication.



Use care when accessing this recessed switch. DO NOT damage adjacent components when changing its position.

16 LEDs

These LEDs show conditions that may be occurring in the controller.

POWER	The power LED indicates there is AC power applied to the drive and the logic supply is active.
BUSY	The busy LED signifies that motion is occurring on the motor.
FAULT	The fault LED indicates an error has occurred in the controller.

17 Motor Wiring

M1	Phase A
M3	Phase A
M4	Phase B
M5	Phase B
Shield	Motor Cable Shield

18 AC Power

AC CONNECTIONS

These inputs connect the single-phase AC power. The input power range is from 90 VAC to 132 VAC.

19 Chassis Ground

This location grounds 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 ataleast as large as the AC supply power wiring.

1.7. Wiring Diagrams



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



This section provides wiring diagrams for each connection. Remember to follow the General Wiring Guidelines.

1.7.1. Motor and Encoder Connections

All motor connections are made via the 5-pin connector. Pin assignments for this connector are:

Controller Connections		Motor Connections	
Pin	**Assignment**	**Leads**	**Terminal**
1	M1 (Phase A)	Red	1
2	M3 (Phase A)	White/Red	3
3	M4 (Phase B)	Black	4
4	M5 (Phase B)	White/Black	5
Shield	Shield	-	-

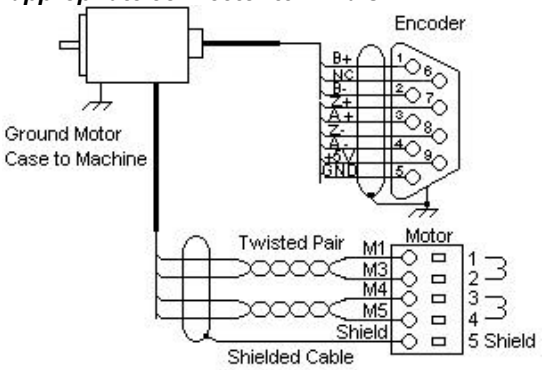
Motor Phase A is M1 and M3. Motor Phase B is M4 and M5. The motor frame MUST be grounded.

Cabling from the drive to the motor should be done with a shielded, twisted pair cable. As a guideline, the wires for each motor phase should be twisted about six times per foot. Danaher Motion Superior Electric offers the following motor cable configurations with unterminated leads on both ends:

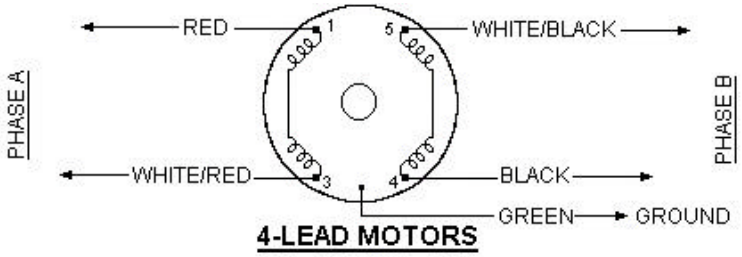
Length	Part Number
10 feet (3 meters)	216022-031
25 feet (7.5 meters)	216022-032
50 feet (15.2 meters)	216022-033
75 feet (22.8 meters)	216022-034

The encoder connections are only required for a closed loop stepper drive (connection scheme depicted below).

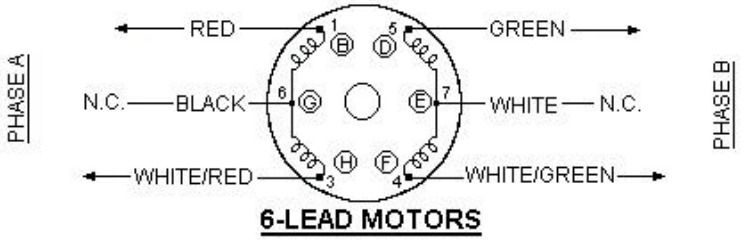
It is IMPORTANT that encoder and motor cables be shielded and the shields be connected to their appropriate connector terminals.



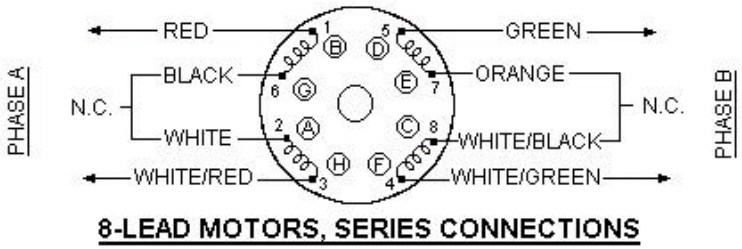
1.7.2. Motor Wiring Configuration



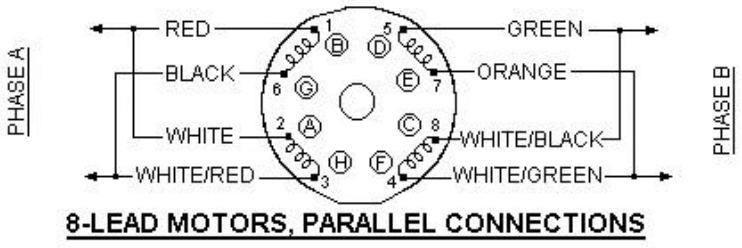
4-LEAD MOTORS



6-LEAD MOTORS



8-LEAD MOTORS, SERIES CONNECTIONS



8-LEAD MOTORS, PARALLEL CONNECTIONS

Colors -- Leaded motors Numbers -- Terminal box motor connections Letters -- Connector motor design

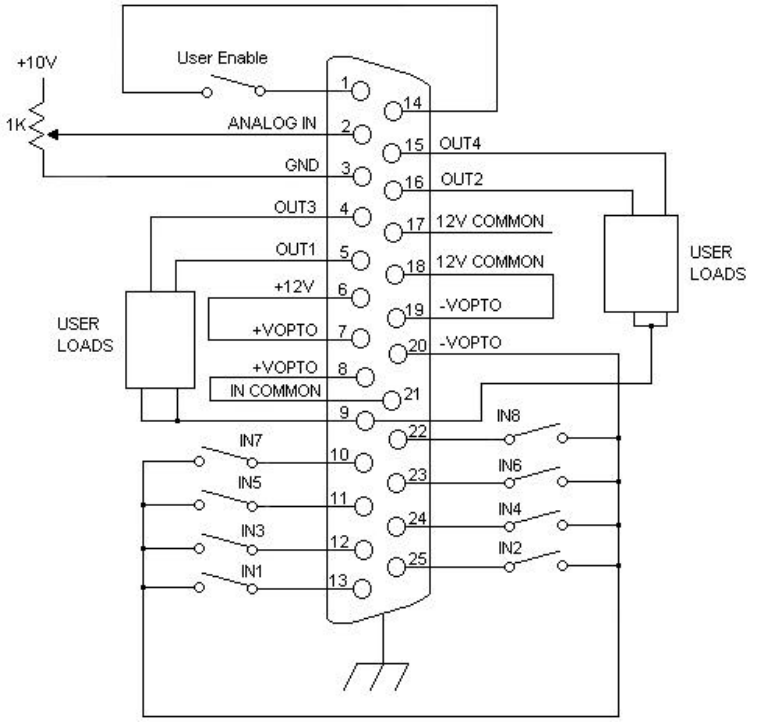
1.7.3. Input/Output Connections

The I/O connections consist of 8 general purpose inputs, 4 general purpose outputs, 1 User Enable input, and 1 analog input. The 8 general purpose input signals can be sinking or sourcing opto-isolated inputs. The input mode (sink or souce) 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 User Enable signal is a sinking input and MUST be connected to GND to enable the stepper motor drive. The analog input has a voltage range from 0 to +10 volts.

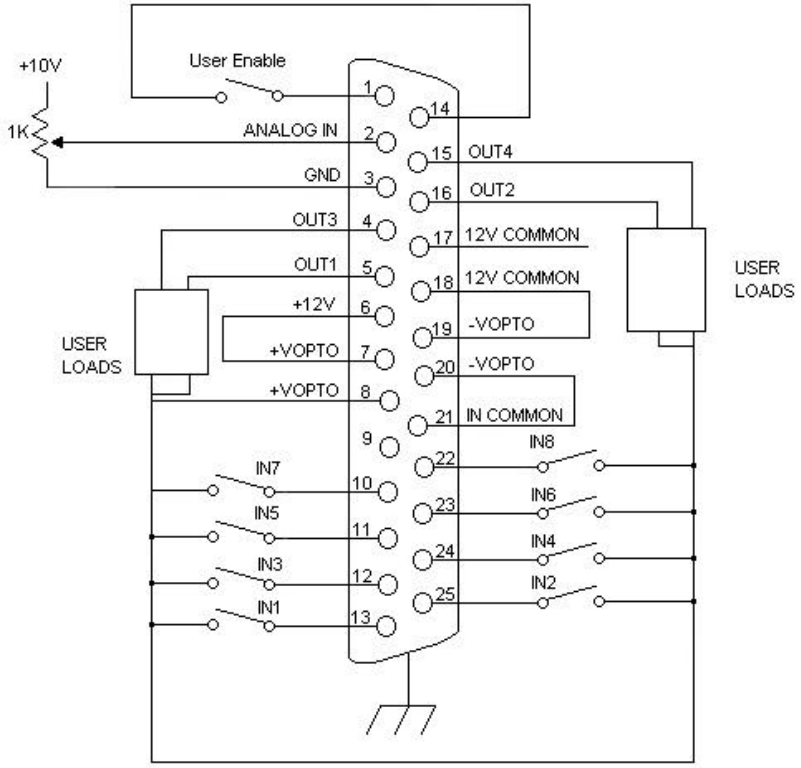
1.7.3.1. INTERNAL POWER SUPPLY

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.

1.7.3.1.1. Inputs Sinking, Outputs Sinking



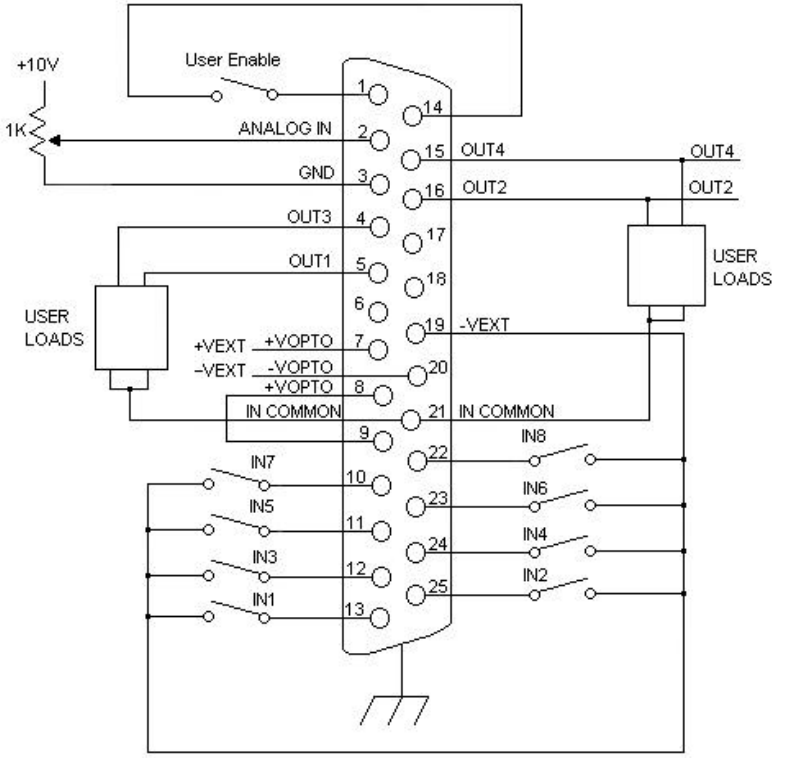
1.7.3.1.2. Inputs Sourcing, Outputs Sinking



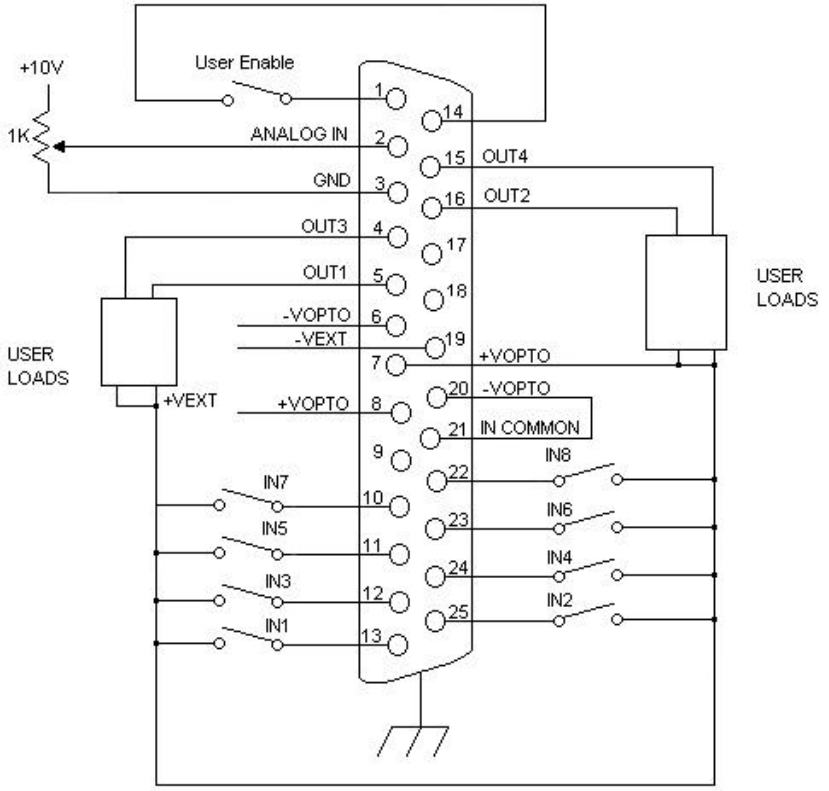
1.7.3.2. EXTERNAL POWER SUPPLY

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 connections are shown for both sinking and sourcing inputs.

1.7.3.2.1. Inputs Sinking, Outputs Sinking

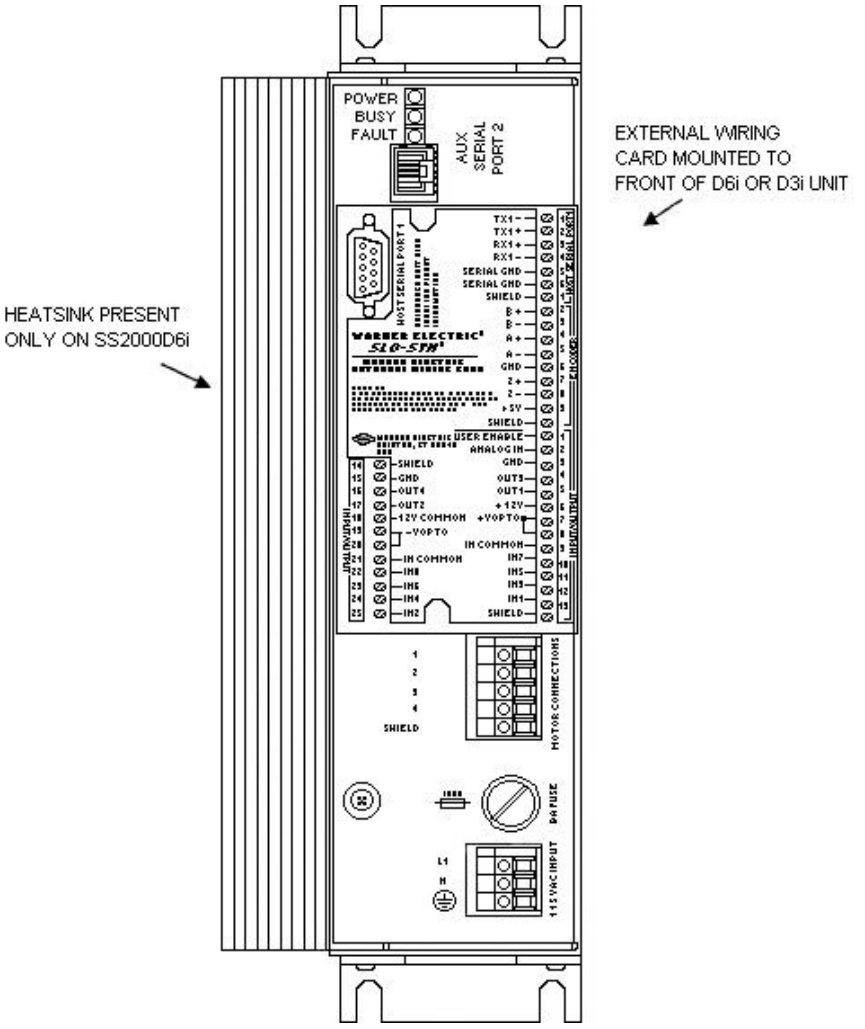


1.7.3.2.2. Inputs Sourcing, Outputs Sinking



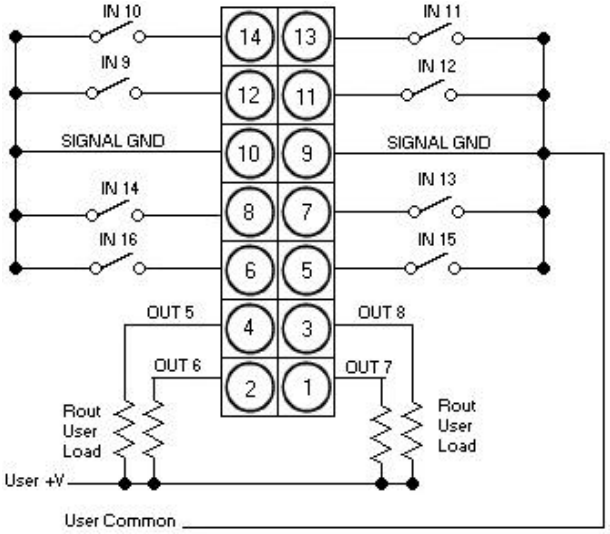
An optional external wiring card is available that provides individual terminals for each I/O point on the "D" style connectors on the D6i or D3i unit. The external wiring card mounts over the D connectors on the face of the unit. All connections are brought out to individual clamp-down terminal connections. The pinouts for the terminal blocks are identical to the D connectors. If you would like to use the optional external wiring card, contact customer support or your distributor and request Part Number XWC.

1.7.4. Optional External Wiring Card



The connections for the BCD I/O connector when used as general purpost signals are depicted in the next figure.

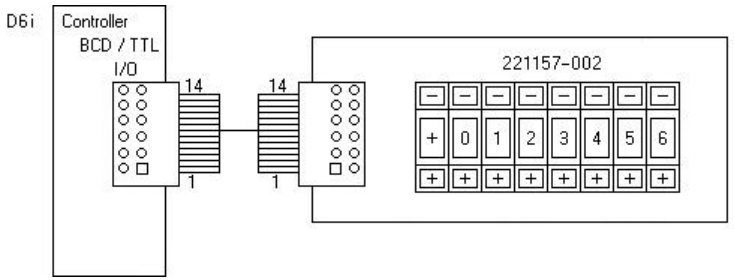
These inputs and outputs are NOT isolated.



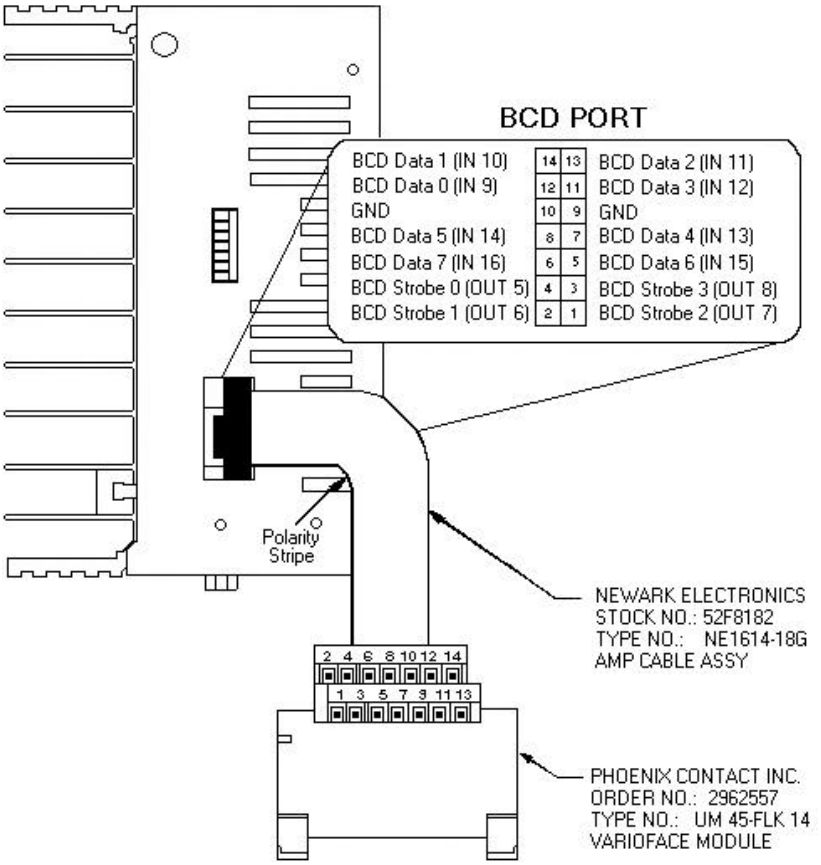
Recommended Output Loads	
User +V	**Rout**
5 VDC	500 Ω
12 to 15 VDC	1.5 kΩ
24 VDC	2.5 kΩ

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

The Danaher Motion Superior Electric BCD switch interface P/N 221157-002 is shown.

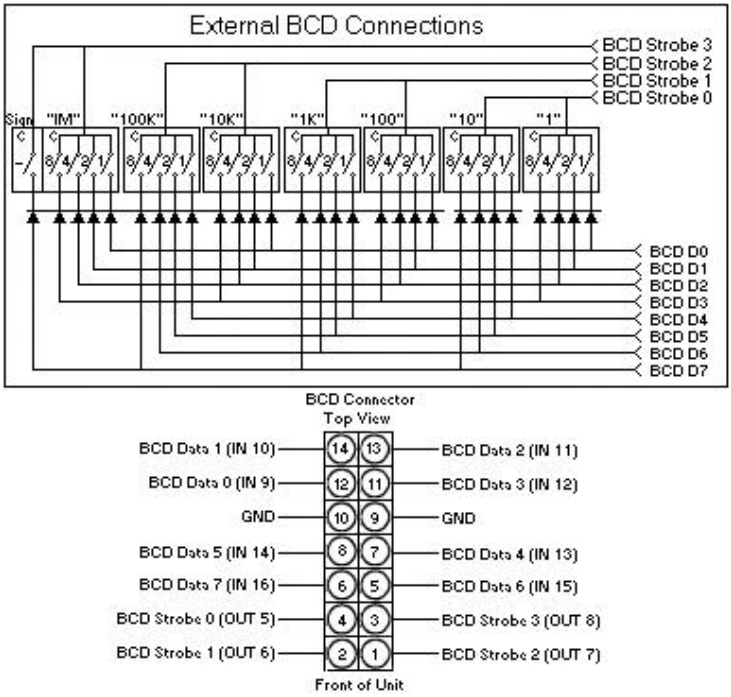


1.7.4.1. USING BCD PORT AS NON-ISOLATED I/O



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

An external BCD configuration is shown in the figure below.

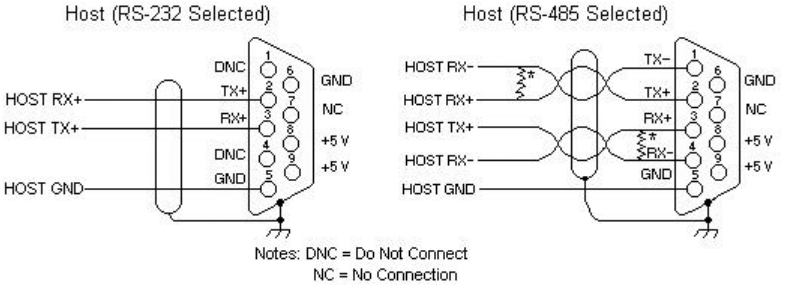


1.7.5. RS-232 / RS-485 Host Serial Communications Connections

This serial port is used for communications and programming of the controller from a personal computer (PC). The port can be configured for either RS-232 or RS-485 operation. A slide switch is provided to make this selection. The factor default is RS-232. The connection diagram for both modes is provided.

When wired for RS-485 operation, a cable with individual twisted pair wires must be used.

The termination resistors indicated with an * must have a value of 120 Ω. If multiple units reside on the RS-485 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 the termination resistor internally. If the terminal device does provide the resistor internally, the resistor across the TX+ and TX- is not required.



1.7.5.1. RS-485 Host Daisy Chaining Connections

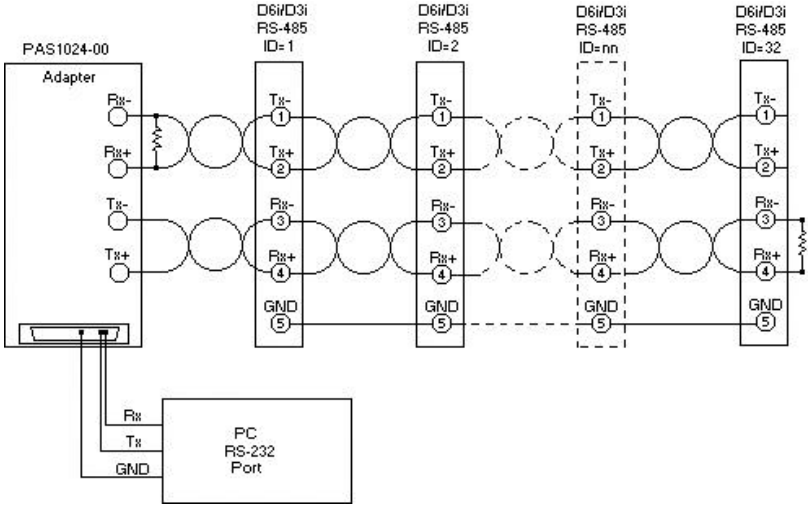
Connection in a daisy chain configuration requires that the Host port of all units be wired as RS-485. Each unit must also be switched to RS-485 Host communications mode by moving the recessed slide switch into the RS-485 position. The switch is accessible through the removable portion of the label on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



Be sure that the unit is off when changing the switch position.



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

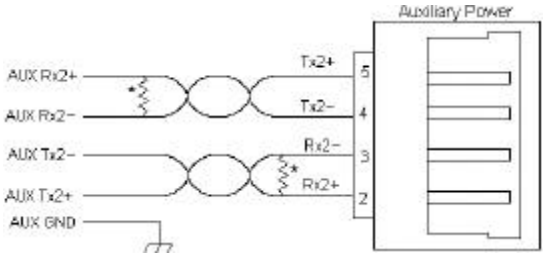


1.7.6. RS-485 Auxiliary Coommunication Connections

The auxiliary serial port is used for serial connections to and from other devices, such as PLCs or operator interface panels. This serial port is RS-485 ONLY and uses a telephone jack for the connections. The wiring connection diagram is shown in the next figure.

A cable with individual twisted pair wires must be used.

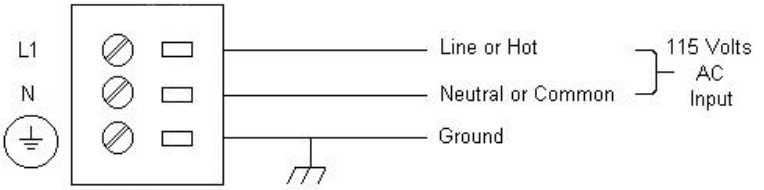
The termination resistors indicated with an * must have a value of 120 Ω. If multiple units reside on the RS-485 bus, ONE resistor should be used at the end of the transmission line bus. The resistor across the Tx+ and Tx- signals may be required if the terminal device does not provide a termination resistor internally. If the terminal device does provide the resistor internally, the resistor across Tx+ and Tx- is not required.



1.7.7. AC Power Connections to the Unit

Connect the two (2) AC IN terminals to the input AC line. The line voltage can be from 90 to 132 VAC 50/60 Hz.

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



Terminal		Lead Color, North American Standard
"L1"	for Line or "Hot"	Black
"N"	for Common or Neutral	White
⏚	for Ground	Green

2. SPECIFICATIONS

2.1. Mechanical

Size

SS2000D6i	
(Inches)	3.67 W x 10.93 H x 5.98 D
(mm)	93.3 W x 277.6 H x 152.0 D
SS2000D3i	
(Inches)	2.33 W x 10.78 H x 5.96 D
(mm)	59.2 W x 273.8 H x 151.8 D

Weight

SS2000D6i	
(pounds)	7.75
(kilograms)	3.53
SS2000D3i	
(pounds)	3.94
(kilograms)	1.70

2.2. Environmental

Operating Temperature	
Fahrenheit	+32° to +122°
Celsius	0° to +50°
Storage Temperature	
Fahrenheit	-40° to +167°
Celsius	-40° to +75°
Humidity	95% maximum, non-condensing
Altitude	10,000 feet (3048 meters) maximum
Pollution Degree	Level 2

2.3. Electrical

AC Input Range	90 to 132 VAC, 50/60 Hz
AC Current	
SS2000D6i	7 A
SS2000D3i	5 A
Fuse Rating **	250 V, 8 A
Fuse Type **	Littlefuse part # 314008 or Bussman part # ABC-8
Drive Power Dissipation	
SS2000D6i	50 watts
SS2000D3i	35 watts

** If this fuse blows, the power supply is prevented from energizing any of its outputs and the unit will not operate. Usually, this fuse only blows if an internal failure occurs. To ensure safety and retain UL rating, the specified rating and type of fuse MUST BE USED.

2.3.1. Isolated Digital I/O

12 VDC I/O Power	11.5 to 14 VDC at 100 mA
Inputs (IN1 to IN8)	
Sink mode (IN COMMON tied to +VOPTO)	
On state voltage range (+Vopto=12 V) with –Vopto=0 V	0 to 6 volts
Input current (VIN=0 V), +Vopto=12 V, -Vopto=0 V	-6 mA
Source mode (IN COMMON tied to –VOPTO=12 V common)	
On state voltage range with –Vopto=0 V	4.5 to 24 V
Input current (VIN=12 V) with -Vopto=0 V	6 mA
Response time (sink or source)	
Opto turn on delay	10 µs (typical)
Opto turn off delay	75 µs (typical)
Programmable Outputs (OUT1 – OUT4)	
Sink mode only	
Continuous current rating	250 mA (max)
Maximum collector voltage with –Vopto=0 V	25 V
On state voltage at 250 mA	1.5 V (max)

2.3.2. Non-Isolated I/O (for BCD Interface)

IN9 -- 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 +5 VDC is provided.

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

OUT5 – OUT8

These are open-collector, sink only outputs that are NOT isolated from the unit's +5 V logic supply. Proper care must be exercised to ensure 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	**0.6 V max at 20 mA**
Permissible output current	**20 mA**
Permissible output voltage	**24 VDC**

2.3.3. Serial Communication

Port 1
Configurable for RS-232 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 to 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 Ω termination resistor. Termination resistors are NOT internal to the unit. The 485 transmitter on the unit is tri-stated when transmission is not occurring.

RS-232

High level output voltage, VOH	5 V min
Low level output voltage, VOL	-5 V max
Input impedance	3 kΩ approximately

RS-485

High level output voltage, VOH	3 V min
Low level output voltage, VOL	0.5 V max
Input impedance	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 Ω termination resistor. Termination resistors are NOT internal to the unit. The 485 transmitter on the unit is tri-stated when transmission is not occurring.

High level output voltage, VOH	3 V min
Low level output voltage, VOL	0.5 V max
Input impedance	High impedance

2.3.4. 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.

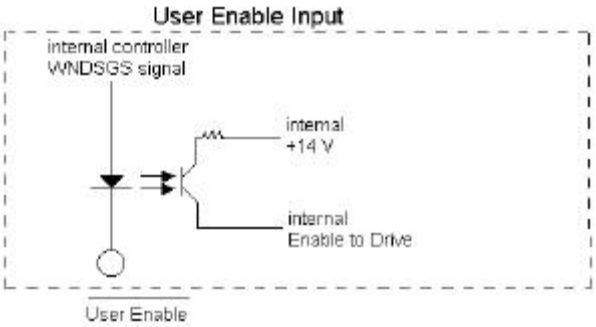
Encoder +5 VDC power supply output	+5 VDC (±5%) at 100 mA current
Encoder signal inputs	TTL level single-ended or differential channels A and B in-phase quadrature
Input current A+, A-, B+, B-, Z+, Z-	±5 mA min
Maximum frequency	500 kHz per channel, 2 megacounts per second in-phase quadrature

2.3.5. Analog Input

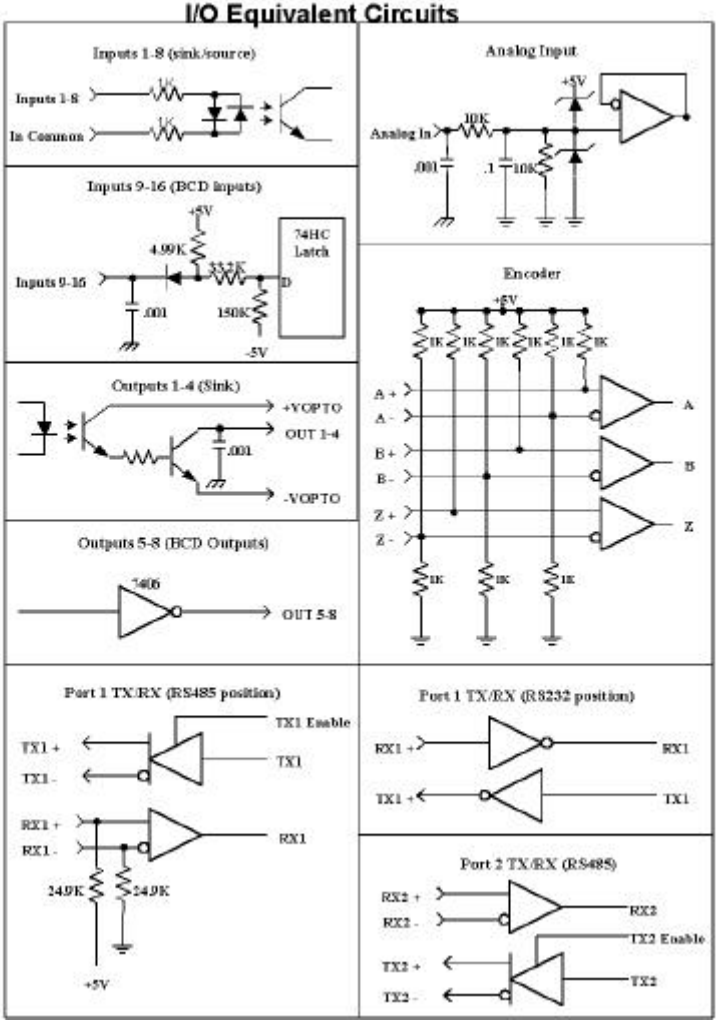
Voltage Range	+10 V (max) to 0 V (min) referenced to GND
Resolution	10 bits or 9.77 mV
Absolute Accuracy	±0.3 V max
Sample Rate	500 Hz min
Bandwidth	100 Hz max
Input Impedance	20 kΩ

2.4. Hardware Equivalent Circuits

The following pages contain equivalent circuit diagrams that 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, Danaher Motion reserves the right to make minor improvements that may change the exact nature of the circuitry, while not degrading functionality.



I/O Equivalent Circuits



2.5. Motor Compatabilty

Motor Types Danaher Motion Superior Electric M and KM Series.
Frame Sizes D3: KML060 — KML091
M061 — M092
D6: KML060 — KML093
M061 — MH112



Do Not use larger frame size motor than those listed, or the drive may be damaged.

Number of Connections 4, 6, 8
Minimum Inductance 8 millihenrys
Maximum Inductance 6 millihenrys
Maximum Resistance 2 ohms at 6 ampere setting



Maximum resistance is total of motor plus cable.

MOTORS FOR USE WITH THE SS2000D6i or SS2000D3i CONTROLLER

M-Series Motors			
P/N	**Current (amperes)**	**D3i**	**D6i**
M061-FF206	1	X	X
M062-FF206	1.5	X	X
M063-FF206	1.5	X	X
M091-FF206	3	X	X
MX91-FF-206U	3	X	X
MX91-FF-206EU	3	X	X
M092-FF206	4		X
MX92-FF-206U	4		X
MX92-FF-206EU	4		X
M093-FF206	4		X
MX93-FF-206U	4		X
MX93-FF-206EU	4		X
M111-FF206	5		X
M112-FF206	6		X
MH112-FF206	6		X

KM Series Motors			
P/N	Current (amperes)	D3i	D6i
KML060-F02	1.5	X	X
KML061-F03	1.5	X	X
KML062-F03	1.5	X	X
KML063-F04	2	X	X
KML091-F05	3	X	X
KLM091-F07	3	X	X
KML092-F07	4		X
KML093-F08	4		X
KML093-F10	6		X

2.6. Motor Performance

All stepper motors exhibit instability at their natural frequency and harmonics of that frequency. Typically, this instability occurs at speeds between 50 and 500 full steps per second and, depending on the dynamic motor load parameters, can cause excessive velocity modulation or improper positioning. This type of instability is represented by the open area at the low end of each motors Torque vs. Speed curve.

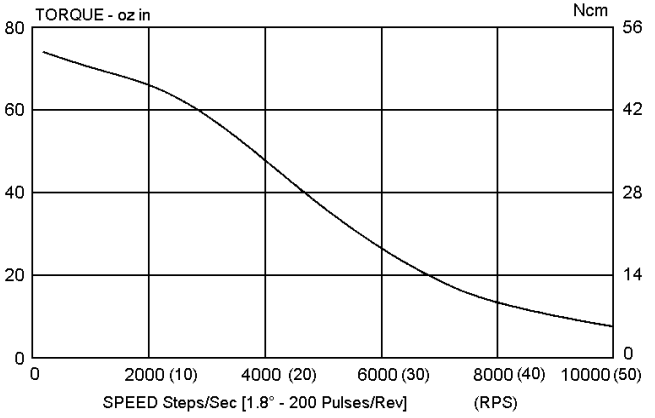
There are also other instabilities which may cause a loss of torque at stepping rates outside the range of natural resonance frequencies. One such instability is broadly defined as mid-range instability. Usually, the damping of the system and acceleration/deceleration through the resonance areas aid in reducing instability to a level that provides smooth shaft velocity and accurate positioning. If instability does cause unacceptable performance under actual operating conditions, the following techniques can be used to reduce velocity modulation.

1) Avoid constant speed operation at the motors unstable frequencies. Select a base speed above the motor's resonant frequencies and adjust acceleration and deceleration to move the motor through unstable regions quickly.
2) The motor winding current can be reduced. Lowering the current proportionally reduces torque. The reduced energy delivered to the motor can decrease velocity modulation.

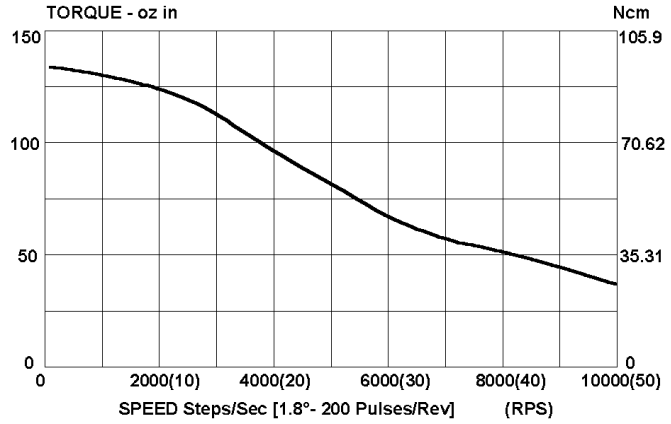
2.7. TYPICAL TORQUE VERSUS SPEED CURVES



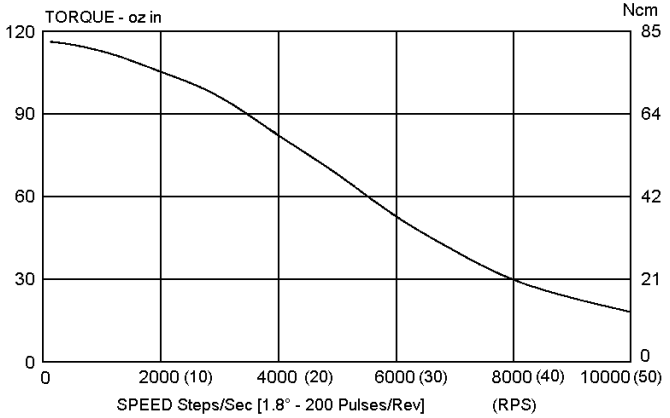
KML060-F02 MOTOR, 1.5 AMPERES



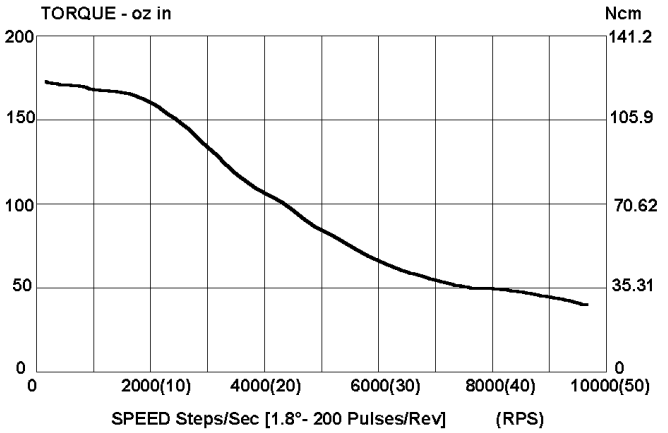
M061-FF206 MOTOR, 1.0 AMPERES



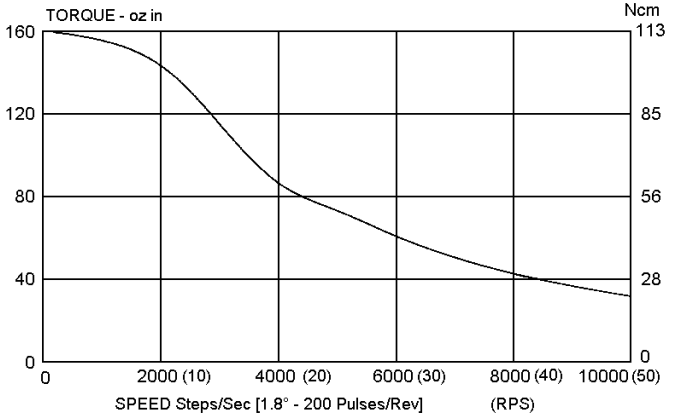
KML061-F03 MOTOR, 1.5 AMPERES



M062-FF206 MOTOR, 1.5 AMPERES



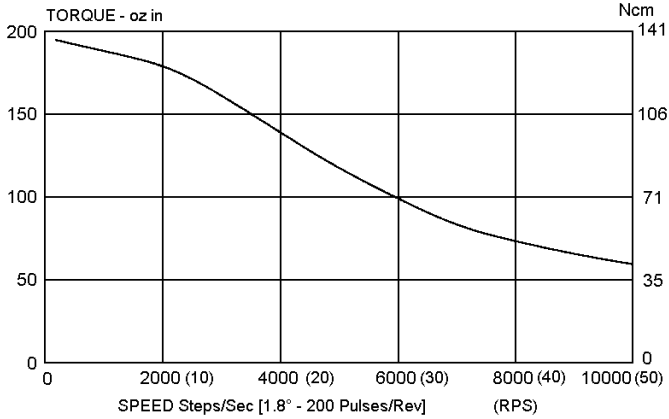
KML062-F03 MOTOR, 1.5 AMPERES



M063-FF206 MOTOR, 1.5 AMPERES



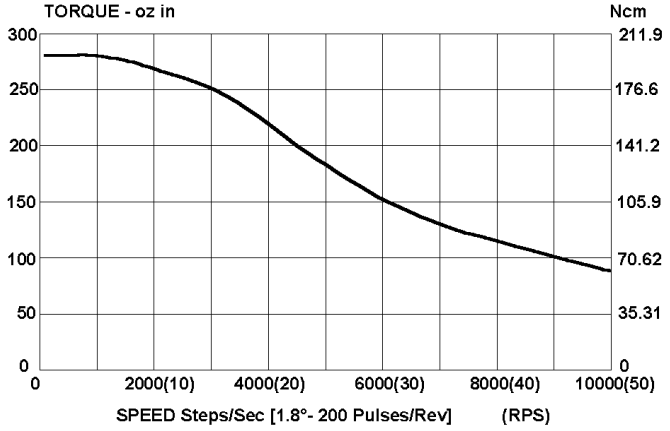
KML063-F04 MOTOR, 2.0 AMPERES



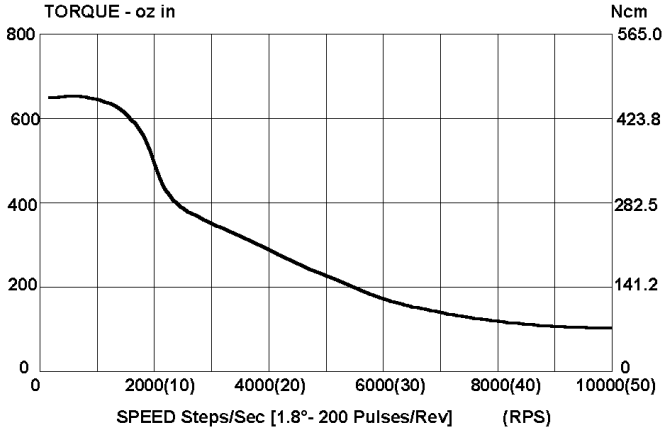
M091-FF206, MX91-FF-206U,
MX91-FF-206EU MOTORS, 3.0 AMPERES



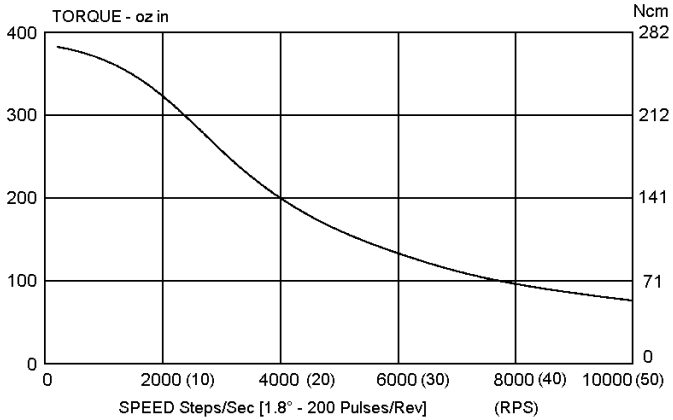
KML091-F05 MOTOR, 3.0 AMPERES



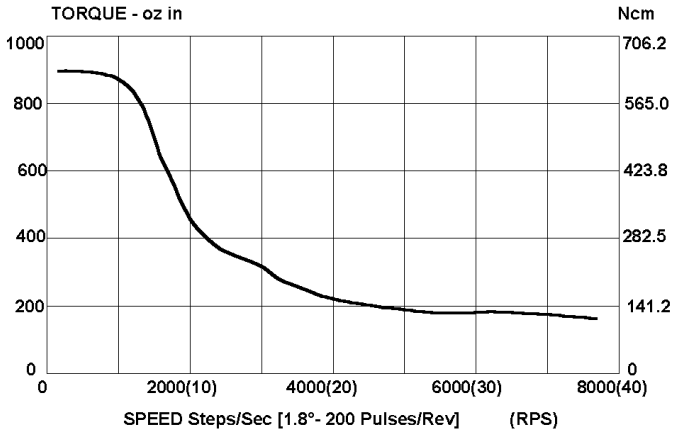
KML091-F07 MOTOR, 3.0 AMPERES



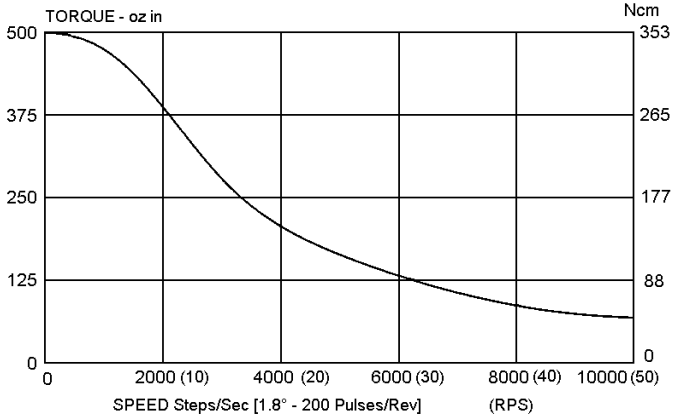
KML092-F07 MOTOR, 4.0 AMPERES



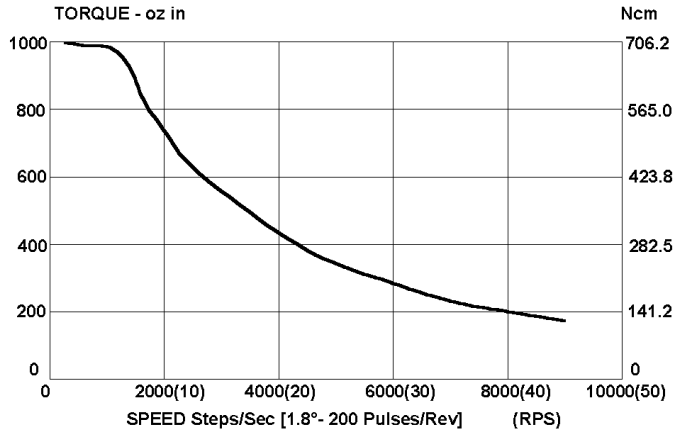
M092-FF206, MX92-FF-206U,
MX92-FF-206EU MOTORS, 4.0 AMPERES



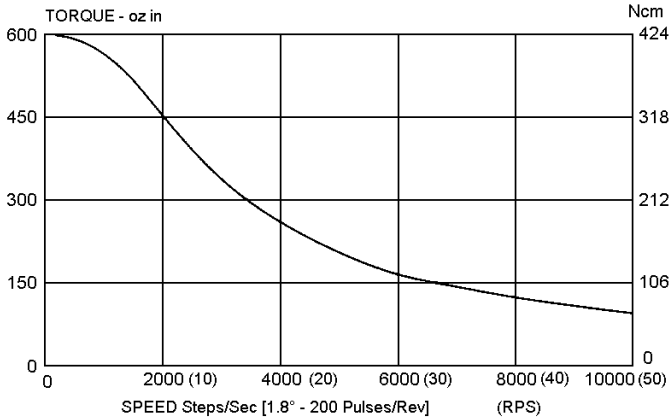
KML093-F08 MOTOR, 4.0 AMPERES



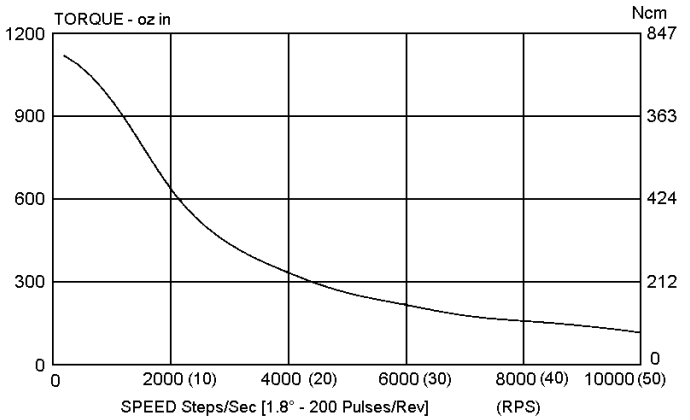
M093-FF206, MX93-FF-206U,
MX93-FF-206EU MOTORS, 4.0 AMPERES



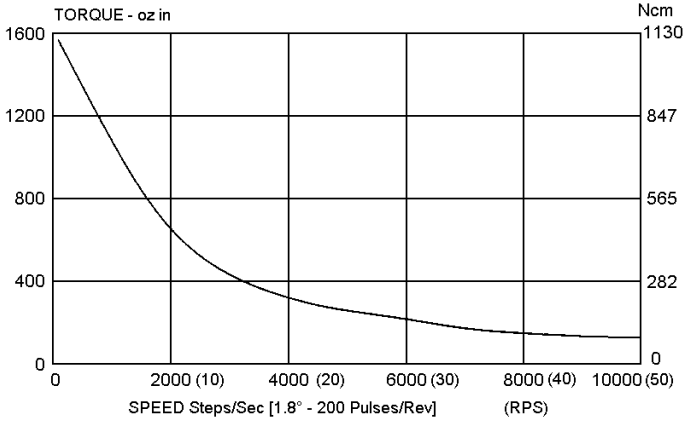
KML093-F10 MOTOR, 6.0 AMPERES



M111-FF206 MOTOR, 5.0 AMPERES



M112-FF206 MOTOR, 6.0 AMPERES



MH112-FF206 MOTOR, 6.0 AMPERES

3. MOTION CONTROLLER PROGRAMMING INTERFACE

All programming requires planning and forethought. This includes programming your controller. This section provides aids to facilitate your planning process.

A program is a list of discrete lines or command strings that, 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.

A program is many individual lines organized in a prescribed sequence. The programming system uses an English language, BASIC-type computer programming language known as Motion Control BASIC or MC-BASIC. This makes it easy and intuitive to write and read machine control programs. MC-BASIC 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.

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.

3.1. Programming the Controller

The programming environment called Motion Controller Programming Interface (MCPI) 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.

3.2. Host Commands

One method of operating the controller is to program it via a PC, then set it up as a stand-alone system. After it is programmed, you do not need to communicate with an outside computer system.

Another method of system operation is connecting the controller to some type of host computer via either the HOST RS-232 or HOST RS-485 port. The computer may direct its operation and query its status from time to time, if desired. Host Commands set or query parameters, start or stop motion, start and stop program execution, etc. You can either use the full command or the abbreviated command (in parentheses).



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, such as "?", are key presses or ASCII characters and are not spelled out in letters. "cr" signifies the carriage return key

3.3. Memory

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

The second kind of memory is **Non-Volatile Memory**, such as FLASH memory, EEPROM or BBRAM memory. Information stored in this type of memory is not lost when power is removed. FLASH memory stores the Operating System and User Program.

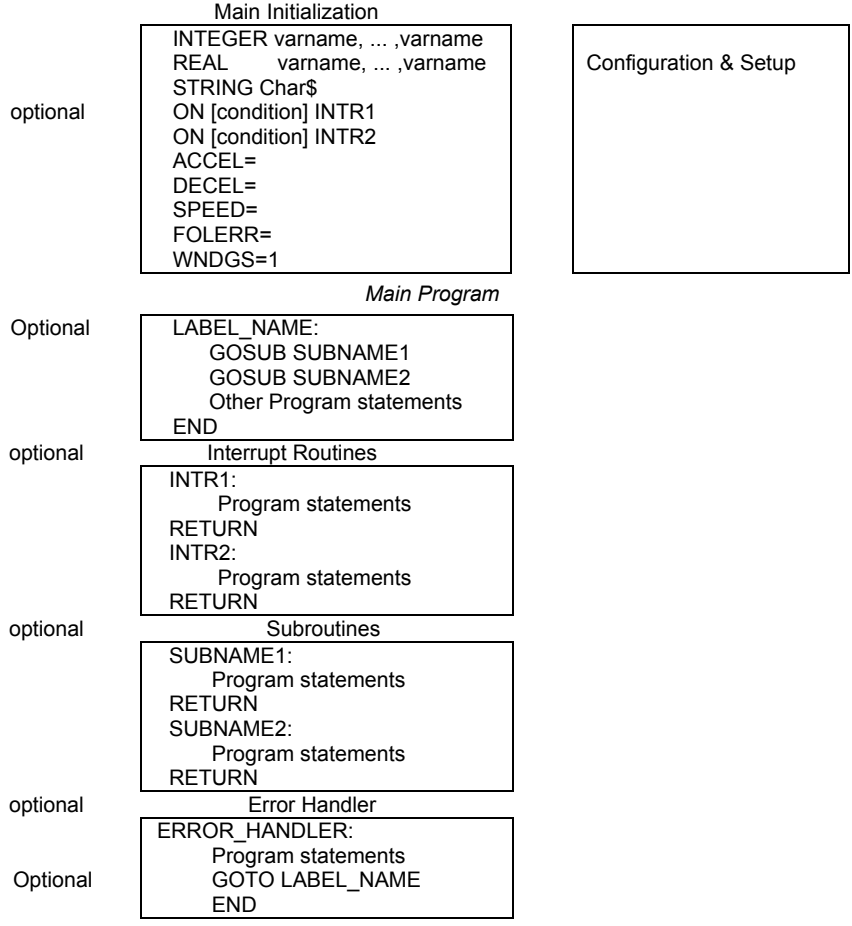
A Controller program can have hundreds of lines of code. Code is simply an organized listing of program commands. Because of the wide variety of program commands it is impossible to state how many lines can be stored in the controller. Use FREEMEM to determine the amount of free memory available.

3.4. Projects

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 a predetermined sequence of instructions.

A good program has 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 the next figure.

Typical Program Development Block Diagram



3.4.1. Initialization

Variable names and data types (Integer, Integer Array, Real and Real Array) are defined in this section. Also the conditions that 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 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
```

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.

3.4.2. Main Program

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

3.4.3. 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 .... INTR*n* 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 INTRON*n* command. If the interrupt condition is true while the interrupt is enabled, then the routine INTR*n* is executed. The INTROFF*n* command disables checking of the interrupt condition. ***n* is a value 1-4**.

3.4.4. Subroutine

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.

3.4.5. 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 terminates when an error condition occurs.

3.5. MOTION CONTROLLER PROGRAMMING INTERFACE (MCPI)

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.

3.5.1. 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

3.5.2. 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.

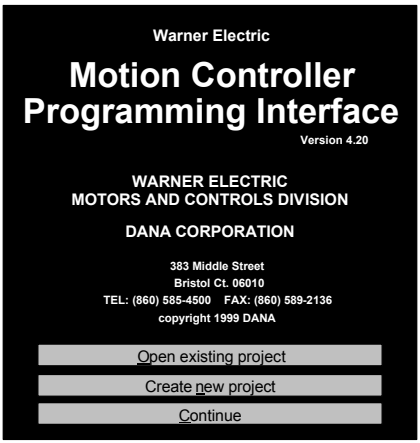
3.5.3. Installation Instructions (Windows 95)

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.

3.5.4. 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.

3.5.4.1. OPENING SCREEN



Open existing project opens up an existing project.

Create new project creates a new project.

Continue enters the MCPI with no selection.

3.5.4.2. 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:

Computer	Controller
TX	*RX+*
RX	TX+
GND	**GND**

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:

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.

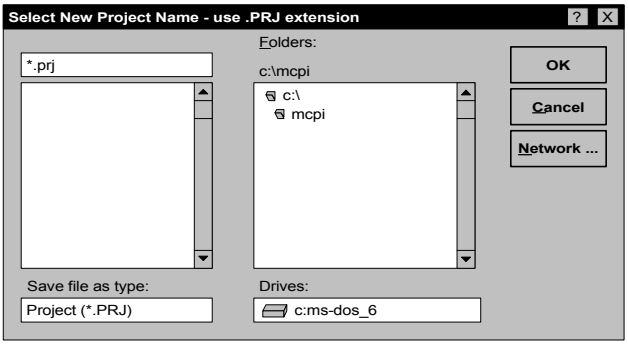
3) Set your terminal to the new baud rate using the System menu and selecting Terminal settings and then Com Port.

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.

Test the serial communications to the controller by clicking **Terminal** and then **Software Revision**. The controller sends the revision information back if the setup is correct.

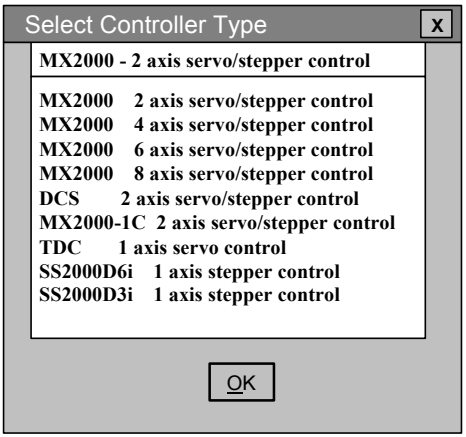
3.5.5. Creating A New Project

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



Enter the name of the project **with a .prj extension**. The directory for the project is also selected at this time. To accept the name and directory, click **OK**.

Select the controller type by clicking on the desired selection and clicking **OK**.



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

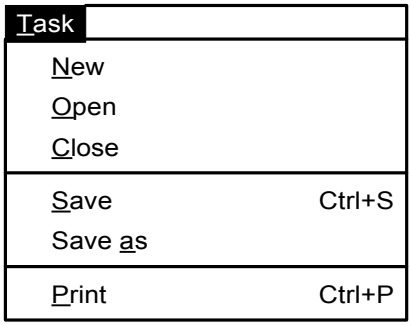


Save each changed folder by clicking **Save changes**. After completing all the changes to the configuration, click **Exit configuration**.

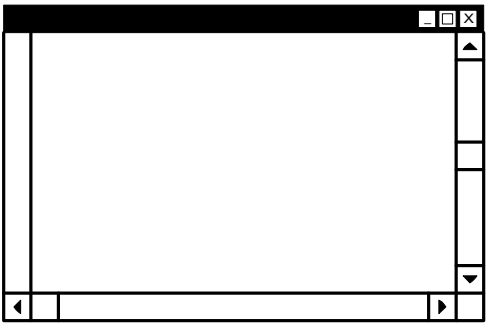
3.5.6. The Task Editor

The Project program is created and edited using the Task Editor. To select the project to be edited, click **Task**, then either the **New** or **Open** item. **New** develops a new task. **Open** edits an existing task.

3.5.6.1. TASK MENU SCREEN

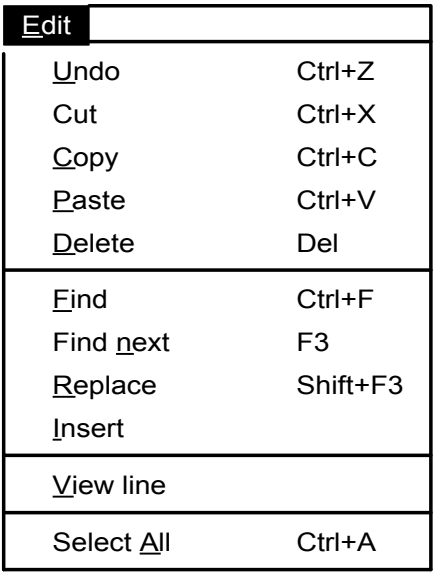


3.5.6.2. TASK EDITOR SCREEN



Access the Edit functions by clicking **Edit** menu and the desired item. The Items and Actions for the **Edit** menu are described below.

3.5.6.3. 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 the **System** menu and then 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.

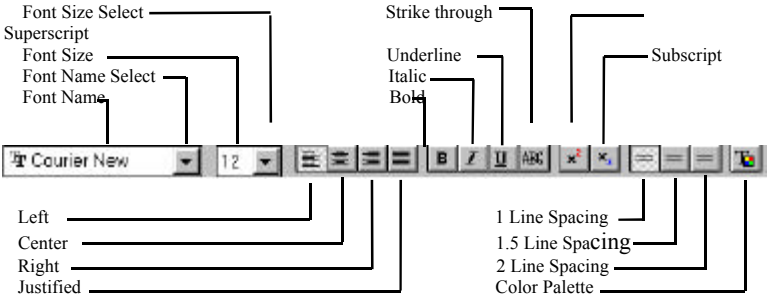
Document format selects the document width, height, margins and tab spacing. Some of these settings are duplicated on the Editor Tool Box.

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

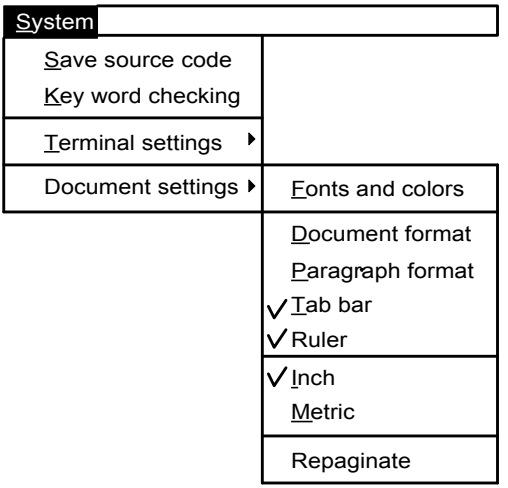
Tab Bar displays the Tab bar when checked.

Ruler displays the ruler when checked.

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.



3.5.6.4. SYSTEM MENU



Inch selects the inch ruler when checked.

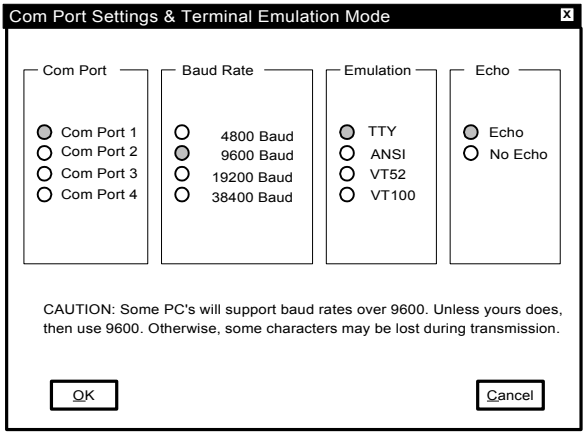
Metric selects the metric ruler when checked.

Repaginate repaginates the current task.

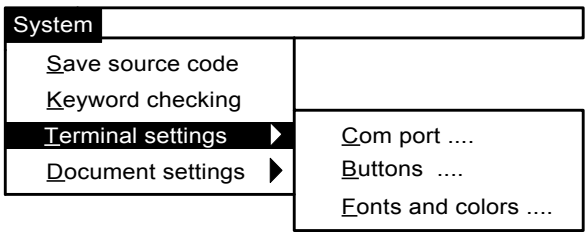
3.5.6.5. TERMINAL EMULATION

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

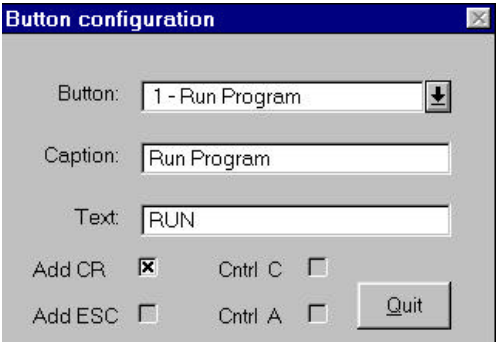
Before entering the Terminal Emulation environment, set up the communication port parameters by clicking on the **System** menu and then the **Terminal settings** item. Choose the appropriate Com port, baud rate, terminal emulation, and echo mode for the Com Port Screen by clicking ON one of the items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



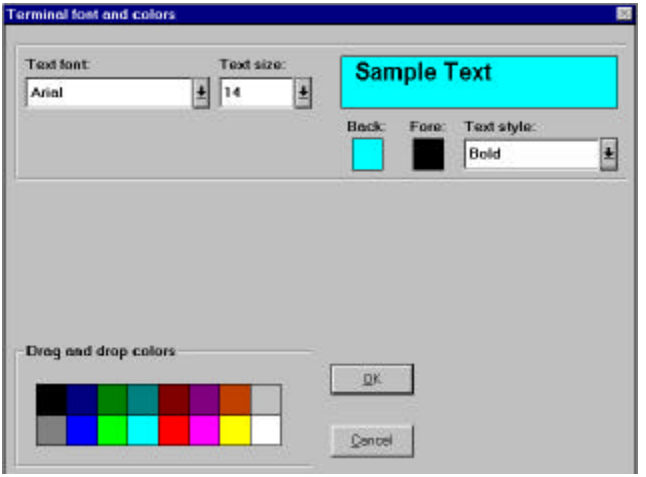
To program the buttons on the Terminal Emulation screen, Click on the **System** Menu and then on the **Terminal settings** item.



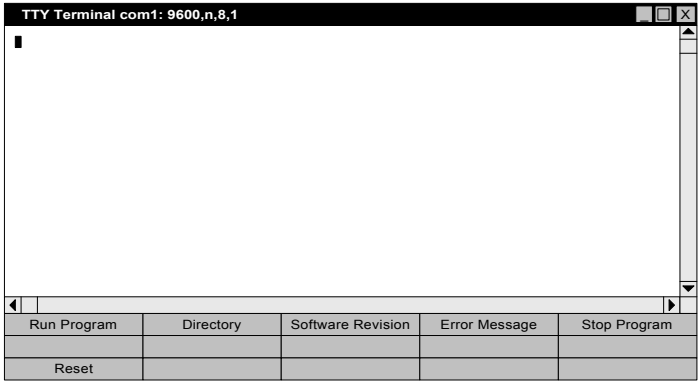
Click the **Buttons** item to open the Button configuration screen. Click on the drop list arrow next to **Button** and select the button number to be programmed. Click on the **Caption** text box and enter the button caption text. Click on **Text** box and enter the command line text to be executed. If motion and program execution is to be stopped after the button's command is executed, click either **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 **Add ESC** check box. Click 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 the **System** Menu and then the **Terminal settings** item. Click 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 O.K.



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



3.5.7. Configuration and Setup Folders

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

Clicking SAVE CHANGES saves the current folder data.

Clicking EXIT CONFIGURATION on any folder exits the Configuration setup. If any of the items in the folder have been changed, a query occur allows you to save the folder data.

Clicking on another folder tab changes to the newly selected folder. The changes already made are not affected. This allows you to click between folders, set up the necessary parameters, and save only once before exiting the configuration screen.

3.5.7.1. SYSTEM FOLDER

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

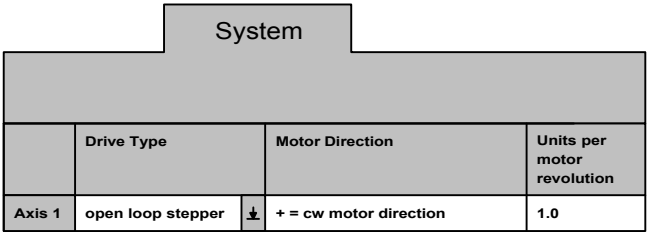
Drive type defines the type of drive operation. The choices are either **open loop stepper** or **closed loop stepper**. Open loop steppers do not have encoders, while closed loop steppers use encoders for position verification and/or correction.

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 "microstep" resolution is fixed at 64 microsteps per full step. This results in 12,800 microsteps per motor revolution for stepper motors with 1.8º full steps.

Enter the desired **Units per motor revolution** value. A unit is the method of measurement (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.



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. Now, you can write your program with distances in inches.

2) Rotary Table
The motor is connected through a 20:1 gearbox to a rotary table. If you wish to program motion in degrees of rotation on the rotary table, 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). A move of 90 results 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 you wish to program motion in feet of movement of the conveyor, 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).

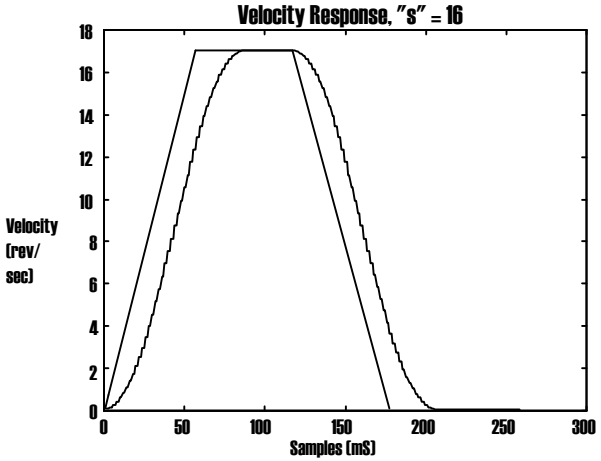
3.5.7.2. PROFILE FOLDER

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 acceleration is applied. There are 32 choices. 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 than those with a setting of 1. The S-curve profiles usually result in smoother motion at the expense of longer move times. Move times can be shortened 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 also decelerates 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.

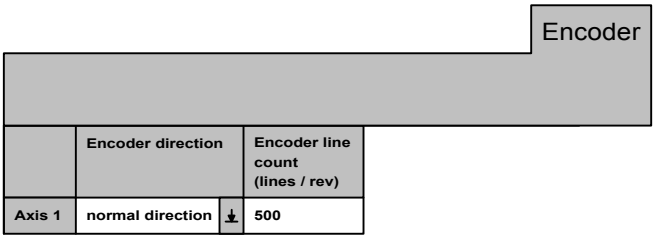
Delay after motion sets the minimum time, in seconds, between two moves.

3.5.7.3. 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.



3.5.7.4. OPEN LOOP STEPPER FOLDER

Sets the Motor standstill current, Motor boost current, Motor current, Drive Current, Smoothing factor and Motor current delay for an open loop stepper drive.

Motor standstill current sets a percentage of motor current when the motor is at standstill. The range of settings are from 0% to 100% in 10% increments.

Motor boost current sets a percentage of motor current when the motor is running. The range of settings are from 100% to 200% in 10% increments.

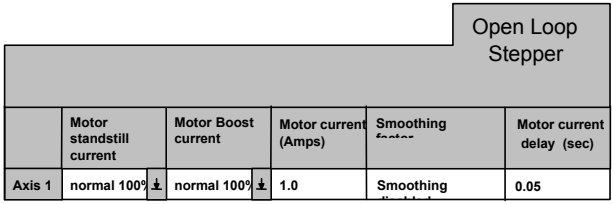
Motor current sets the maximum peak current in amps of the motor. This corresponds to the 100% setting in the standstill and boost modes.



This number must be set at or below the nameplate current rating of the motor or motor overheating and damage may result.

Smoothing factor allows different profile of smoothing to be applied to the motor current when moving. The selection are**: smoothing disabled**, **smoothing profile 1**, **smoothing profile 2** and **smoothing profile 3**. Smoothing reduces motor vibration at slow speeds. Different motors and loads may have different smoothing settings.

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



3.5.7.5. CLOSED LOOP STEPPER FOLDER

This folder sets the **Motor standstill current, Motor boost current, Motor current, Drive Current, Smoothing factor, Motor current delay, Error Action, Correction attempt**s and **Time between attempts** for a closed loop stepper drive.

Motor standstill current sets a percentage of motor current when the motor is at standstill. The range of settings are from 0% to 100% in 10% increments.

Motor boost current sets a percentage of motor current when the motor is running. The range of settings are from 100% to 200% in 10% increments.

Motor current sets the maximum peak current in amps of the motor. This corresponds to the 100% setting in the standstill and boost modes.



This number must be set at or below the nameplate current rating of the motor or motor overheating and damage may result.

Smoothing factor allows different profile of smoothing to be applied to the motor current when moving. The selection are**: smoothing disabled**, **smoothing profile 1**, **smoothing profile 2** and **smoothing profile 3**. Smoothing reduces motor vibration at slow speeds. Different motors and loads may have different smoothing settings.

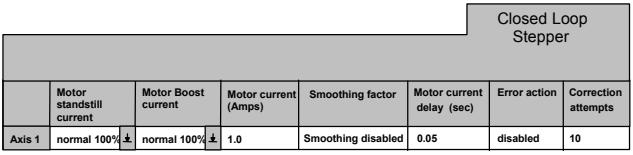
Motor current delay specifies the time delay between current modes in seconds. This allows time for the drive to respond to the change in current level as a result of either BOOST or REDUCE (see Program Command section).

Error action. This setting 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:

1) If **Error action** is **disabled**, the controller takes no action.
2) If **Error action** is **stop on error**, the motor stops and a controller error results (see ERR command). The fault light illuminates.
3) If **Error action** is **correct on error**, separate correction attempts (moves) are commanded to re-align the motor. You may specify **how many correction** attempts 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 results.
4) 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 again returns to the start position and retries 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.

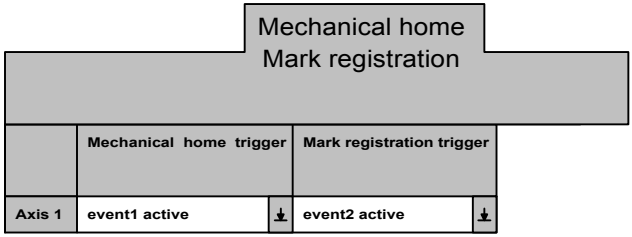
Correction attempts specifies the maximum number of consecutive attempts allowed when **error action** is set to **correct on error** or **restart on error** mode and the motor stalls.

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.



3.5.7.6. 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 that 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
Event 2 Inactive

3.5.7.7. I/O FOLDER

This folder assigns inputs 3-7 as either 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)** are configured as either hard limit inputs or general purpose inputs. As limits, the active signal level is also configured as either active on switch closing or active on switch opening.

Inputs 5, 6 & 7 are *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 starts program execution unless the **Clear** input is inactive. Following the initial power up, an inactive-to-active transistion on the **Run** input does the following:

If the **Feedhold** state is set, and the **Feedhold** input is inactive, **Run** clears the feedhold state.

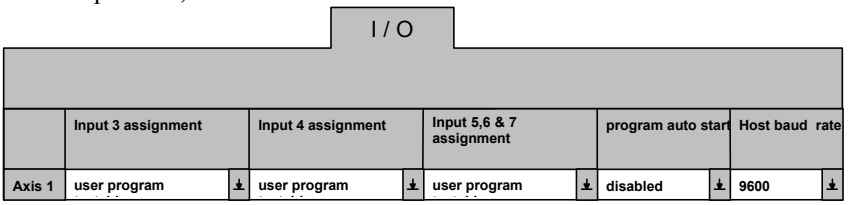
If the **Clear** input is active, **Run** simply starts the user program.

Clear- An **inactive** clear input stops 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 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 Run, 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. 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 is 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 takes effect upon power up or reset. If the baud rate switch is in the 9600 position, all communications occur at 9600 Baud.



3.5.8. Preparing User Project for Execution

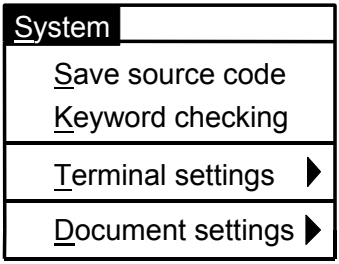
To execute a project program, it must first be compiled and downloaded to the controller. The project source code can be recovered from the controller.

3.5.8.1. PROJECT SOURCE CODE

The Project Source Code is the English version of the user's program. If the program needs to be uploaded from the controller at any time, **Save Source Code** must be enabled. The Source code of a project is 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 **System**. The Save source code setting is toggled by clicking the **Save source code** item.

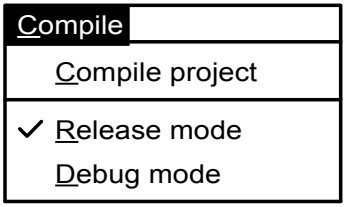


Saving source code in the controller requires a lot of program memory. If your program is extremely long, it may not be possible to save the source code. See FREEMEM for more information.



3.5.8.2. 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 the Debug Environment).



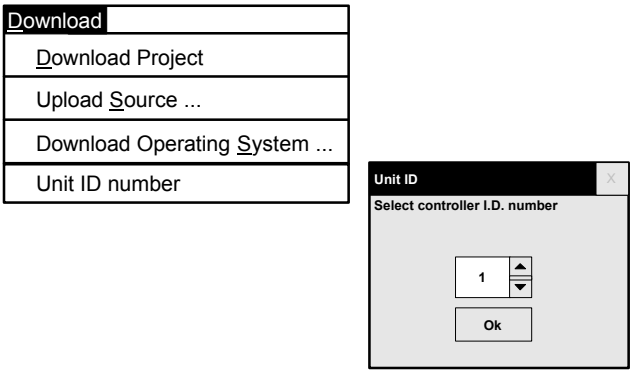
To set the Debug mode, click **Compile** and **Debug mode**. The project must now be compiled and downloaded before task debugging can begin. To cancel the debugging mode, click **Compile** and **Release mode**. To complete this cancellation, the project must now be compiled and downloaded.

3.5.8.3. 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 user tasks and configuration to machine code that the controller can understand. A project can be compiled by clicking the **Compile** button or **Compile** and **Compile project**.

3.5.8.4. SELECTING THE CONTROLLER UNIT ID

The controller unit ID is selected by clicking **Download** and **Unit ID number**. The unit ID can now be selected using the spin controller and clicking **OK**.

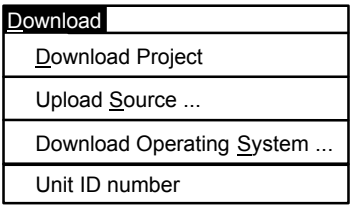


3.5.8.5. DOWNLOADING A PROJECT

A project is downloaded to an active unit either with or without its source code by clicking the **Download** button or clicking **Download** and **Download project**. The project is sent only to the active unit.

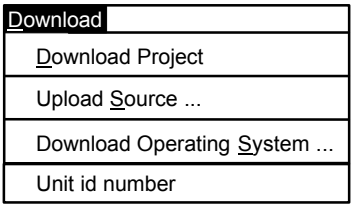
3.5.8.6. UPLOADING SOURCE CODE

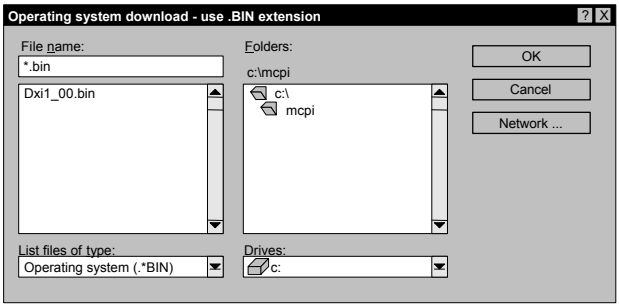
A project's source code is uploaded from the controller to the PC by selecting **Download** and **Upload Source**. A project is uploaded from the controller ONLY if it had previously been saved in the controller. The upload is from the active controller when there is more than one unit on a daisychain.



3.5.9. Downloading an Operating System

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





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



The file names for the different controllers start with the following letters:

mx for the MX2000 controller
dcs for the DCS controller
tdc for the TDC controller
DxI for the SS2000D6i/D3i controllers

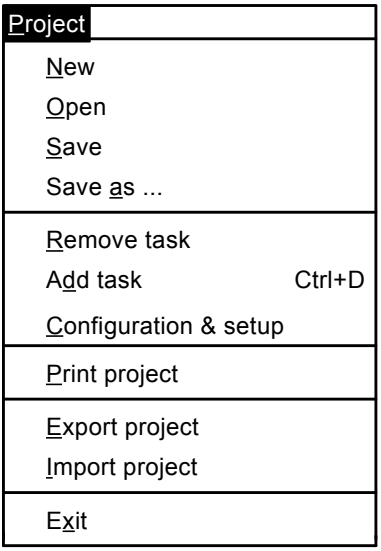
If on a daisychain, the appropriate unit ID number must first be selected.

3.5.10. 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 simutaneously.

3.5.10.1. 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 and setup environment, print a current project, or exit the MCPI programming environment.



New creates a new project.

Open opens up an existing Project.

Save saves the current project.

Save as saves the current project under a new name.

Remove task removes a task file from an open project.

Add task adds a file to a current project.

Configuration & setup allows editing of the Configuration and setup folders.

Print project prints a current project's information.

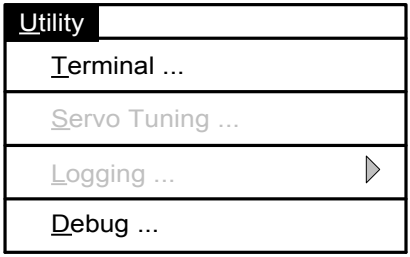
Export project exports a current project to another drive or directory.

Import project imports a selected project from another drive or directory into the MCPI Environment.

Exit exits the MCPI programming Environment.

3.5.10.2. 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.

3.5.10.3. WINDOW MENU

This menu selects the windows format for the open windows.

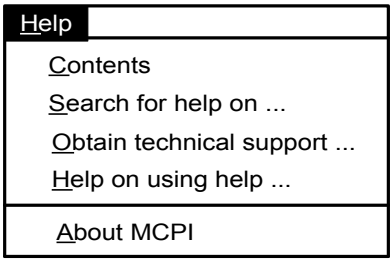
Cascade cascades the open windows.

Tile Horizontal tiles the open windows Horizontally.

Tile Vertical tiles the open windows Vertically.

3.5.10.4. 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.

3.5.11. 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	Compile project	Download project	Terminal	Servo tuning	Debug

Configuration enters the configuration & setup environment.

Compile project compiles a current project.

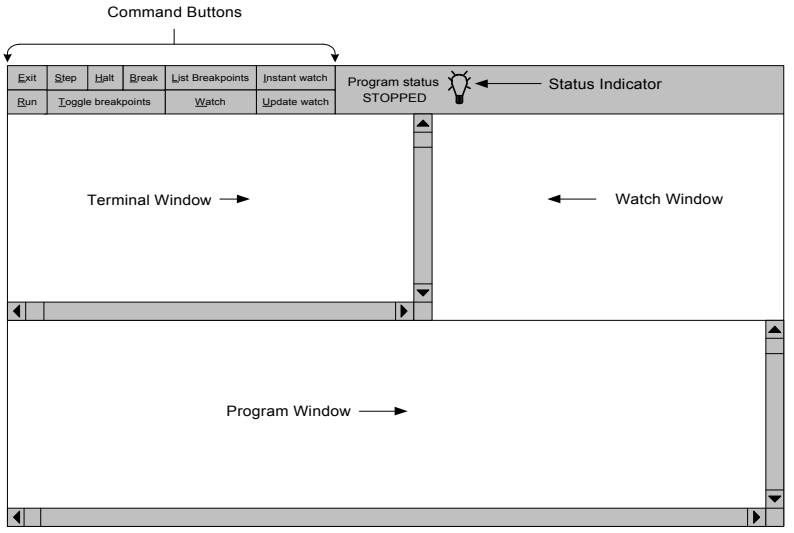
Download project downloads a current project.

Terminal enters the Terminal emulation environment.

Debug enters the Program Debugger Environment.

3.5.12. DEBUG Environment

A project loaded in the controller can be debugged if the project has been compiled in Debug mode and downloaded. The project to be debugged must be open. To enter the debugger environment, click **Debug**. 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.



3.5.12.1. DEBUG PROGRAM EXECUTION

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



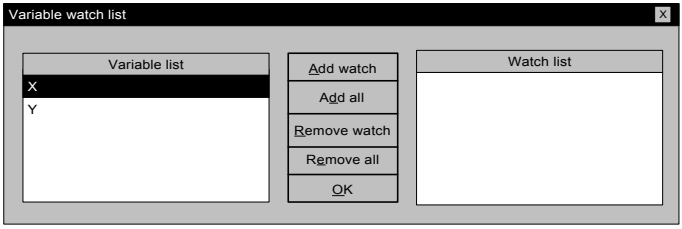
The program status indicator shows the status of program execution. The only time this status indicates Stopped is when the program is halted or has executed an end statement in the program. The indicator is green for running and red for stopped.

3.5.12.2. BREAKPOINT SETTING/CLEARING

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

3.5.12.3. VARIABLE WATCH

To view the values of selected variables, use the Variable watch. To add or remove a watch variable from the watch window, click **Watch**.

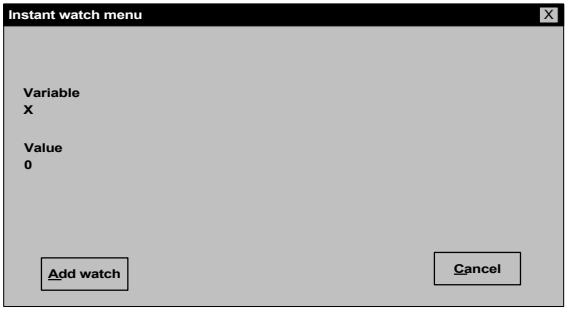


To add all the variables to the watch list, click **Add all**. To add a specific variable to the watch list, select the variable in the Variable list and click **Add watch**.

To remove all variables from the watch list, click **Remove all**. To remove a specific variable from the watch list, select the variable in the Watch list and click **Remove watch**.

To return to the Debug Environment screen, click **OK**. The variable in the watch list appears in the Watch Window and its current value is displayed.

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



3.5.12.4. 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 the Terminal window is selected for host commands.

3.5.12.5. EXIT DEBUG ENVIRONMENT

Exit the debug environment by clicking **Exit**.

4. SOFTWARE REFERENCE

4.1. Programming Commands By Function

4.1.1. Arithmetic Operators

+	addition
-	subtraction or unary minus
*	multiplication
/	division

4.1.2. Boolean Operators

AND	Logical conjunction operator.
NOT	Logical complement operator.
OR	Logical inclusive OR operator.

4.1.3. Interrupt

INTROFF*n*	Disable interrupt *n*, where *n* is 1-4.
INTRON*n*	Enable interrupt *n*, where *n* is 1-4.
ON...INTR*n*	On condition go to interrupt *n*, where *n* is 1-4.

4.1.4. I/O

ANALOG	Returns the analog input voltage.
BCD	Returns the BCD switch value.
IN	Returns the discrete input state of the defined input.
OUT	Sets or returns the discrete output state of the defined output.

4.1.5. Miscellaneous

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

4.1.6. Motion

BOOST	Enables or disables the Boost Current feature of a stepper or returns the boost status.
BUSY	Returns the motion status of the axis.
EVENT1	Sets Enable/disable and trigger state of event1.
EVENT2	Sets Enable/disable and trigger state of event2.
JOG	Run continuously in the specified direction.
MOVEA	Initiates an absolute indexed move.
MOVEHOME	Run until the home input is activated.
MOVEI	Initiates an incremental indexed move.
MOVEREG	Run until the registration input is activated, then move the specified distance.
REDUCE	Enables or disables the Reduce current feature of a stepper or returns the Reduce status.
STOP	Brings any motion to a controlled stop.
STOPERR	Sets or returns the maximum position error allowed when motion is stopped.
WAITDONE	Waits for motion to be done.
WNDGS	Enable/Disable drive.

4.1.7. Over Travel Limit

HARDLIMOFF	Disables hard limits.
HARDLIMON	Enables hard limits.
REGLIMIT	Sets or returns the movereg limit distance.
SOFTLIMNEG	Sets or returns the absolute negative travel limit position.
SOFTLIMOFF	Disables soft limits.
SOFTLIMON	Enables soft limits.
SOFTLIMPOS	Sets or returns the absolute positive travel limit position.

4.1.8. Program Flow Control

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

4.1.9. Relational Operators

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

4.1.10. String Manipulation

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

4.1.11. Time Functions

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

4.1.12. Trajectory Parameters

ABSPOS	Sets or returns the absolute position.
ACCEL	Sets or returns the acceleration rate in units/sec/sec.
DECEL	Sets or returns the deceleration rate in units/sec/sec.
DIST	Returns the distance moved from the start of the last commanded motion or changes the move distance during indexed (MOVEA, MOVEI) motion.
ENCPOS	Returns the encoder absolute position.
ENCSPD	Returns the current speed.
FOLERR	Sets or returns the position error limit for a closed loop stepper.
LOWSPD	Sets or return the starting speed value of a stepping motor.
SPEED	Sets or returns the commanded target speed.

4.1.13. Variable Definitions

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

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

4.2. Host Commands By Function

4.2.1. MOTION

BUSY (BS)	Returns the motion status of the axis.
EVENT1 (E1)	Sets enable or disable and trigger state or returns event1 input state.
EVENT2 (E2)	Sets enable or disable and trigger state or returns event2 input state.
FEEDHOLD (FH)	Control stops any motion.
JOG (J)	Runs continuously in the specified direction.
MOVEA (MA)	Initiates an absolute indexed move .
MOVEHOME (MH)	Run until the home input is activated.
MOVEI (MI)	Initiates an incremental indexed move.
MOVEREG (MR)	Runs until the registration input is activated, then move the specified distance.
STOP (S)	Control stop any motion.
STOPERR	Sets or returns the maximum position error allowed when motion is stopped.
WNDGS (WN)	Enable/disable drive.

4.2.2. TRAJECTORY PARAMETERS

ABSPOS (P)	Sets or returns the absolute position.
ACCEL (AC)	Sets or returns the acceleration rate in units/sec/sec.
DECEL (DC)	Sets or returns the deceleration rate in units/sec/sec.
DIST	Returns the distance moved from the start of the last commandedmotion or changes the move distance during indexed motion.
ENCPOS (EP)	Returns the encoder absolute position.
ENCSPD (ES)	Returns the current speed.
FOLERR (FE)	Sets or returns the position error limit for a closed loop stepper.
LOWSPD	Sets or return the starting speed value of a stepping motor.
SPEED (SPD)	Sets or returns the commanded target speed.

4.2.3. I/O

ANALOG (AN)	Returns the analog input voltage.
BCD	Returns the BCD switches value.
IN (I)	Returns the discrete input state of the specified input.
OUT (O)	Sets the discrete output state of the specified output.

4.2.4. TRAVEL LIMITS

HARDLIMOFF (HL0)	Disables hard limits.
HARDLIMON (HL1)	Enables hard limits.
REGLIMIT (RL)	Sets or returns the MOVEREG limit distance.
SOFTLIMNEG (SLN)	Sets or returns the software absolute negative travel distance.
SOFTLIMOFF (SL0)	Disable soft limits.
SOFTLIMON (SL1)	Enables soft limits
SOFTLIMPOS (SLP)	Sets or returns the software absolute positive travel distance.

4.2.5. MISCELLANEOUS

CURRENT	Returns the project drive current setting.
DIR	Returns the user project information.
ERR	Return error number.
ERRM	Return error number.
FREEMEM	Return the user program free memory byte value.
RESET	Reset operating system.
REVISION (REV)	Returns operating system revision and date.
RUN	Execute user program from the start.

4.2.6. IMMEDIATE

“BACKSPACE”	Delete one character in the host buffer.
“CTRL-A”	Stop motion immediately and terminate program execution.
“CTRL-C”	Stop motion immediately and terminate program execution.
“ESCAPE”	Immediate Host command.

4.2.7. DAISY CHAINING

<nn	Enables a specific unit (nn = 01 to 32)
<nn?	Enables a specific unit (nn = 01 to 32) which then sends back its unit ID.
<00	Places all units on the chain in a command listen mode, the units cannot transmit.

4.3. MC-BASIC Conventions

A BASIC-like language (Motion Control BASIC or MC-BASIC) that conforms to most of the rules and conventions of the BASIC programming language, such as "QuickBasic", etc. The following is a summary of the considerations to be used when writing your programs.

4.3.1. Arithmetic Operators

The MC-BASIC 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

*Squaring and exponentiation are not supported. Use multiplication to perform these operations. For example: to calculate X3, use X*X*X.*

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.

4.3.2. Logical Operators

These operators are used in boolean expressions. The logical operators in MC-BASIC, listed in order of precedence are:

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 for use in making a decision.

4.3.3. Relational Operators

Relational operators compare two values. The result of the comparison is either true (one) or false (zero). This result is then 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.*

4.3.4. 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

Either "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 the Glossary.

4.3.4.1. 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 INTEGERs, the operation produces 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 is 2 because num1 and denom1 were declared as INTEGER numbers. The value of answer2 is 2.5, as expected because denom2 was declared as a REAL variable. When assigning a variable to a number 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 is equal to 2 since 10 and 4 without a decimal point are integers
x = 10.0/4  'x is equal to 2.5 because of the decimal point in 10.0
```

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

4.3.4.2. CASE SENSITIVITY

Some programming statements and commands are case-sensitive. Others are not. The following table defines case-sensitivity in MC-BASIC:

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

Host commands are *not* case sensitive. Upper- and lower-case letters can be used interchangeably.

4.3.4.3. CALCULATIONS USING TRAJECTORY PARAMETERS AND VARIABLES

Use caution 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 because 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 you are programming in units, the actual position is calculated based on the number of microsteps per user unit.

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

4.3.4.4. 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 apostrophe to the end of line is not considered part of the command during execution.

EXAMPLE:

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

4.4. Programming Commands

"Backspace"

Immediate

Host Syntax Press the BACKSPACE key or send ASCII 08.

Description Use either the Backspace key or ASCII code 08 to delete one character from the host receiver buffer.



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

"Ctrl-A"

Immediate

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

Description Stops program execution and 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.



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

"Ctrl-C"

Immediate

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

Description Stops program execution and 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.



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

"Escape"

Immediate

Host Syntax Press the ESCAPE key or send ASCII code 27.

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



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

Example

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

<nn

Daisy Chain

Host Syntax **<*nn* cr**

Description Enables a specific unit on the Host daisy chain to receive and transmit information. *nn* is a unit ID number from 01 to 32. Leading zeros are required when specifying unit ID numbers less than 10. 00 is a special case.

This Host command communicates 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. 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 transmits a RDY upon reset.

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 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 sends a <*nn*? cr . If any unit is on the chain and in the active mode, it transmits its ID number as two characters.

All controllers on the chain may be placed in a command listen mode. In this mode, all units actively listen for and respond to commands, but do 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. To exit the command listen mode, an individual unit must be re-activated (e.g. <01).

Example

```
<05      ‘ sets unit with ID number 5 to the active mode
<06?     ‘ queries whether unit 6 is on the chain and active
         ‘ unit 6 responds with “06” if it is
<00      ‘ sets all units to the command listen mode
```

ABSPOS

Trajectory Parameter

Program Syntax **ABSPOS=*expression***
ABSPOS - used in an expression

Host Syntax **ABSPOS=*number* cr**
ABSPOS cr

Abbreviation **P**

Description Sets or returns the commanded absolute position of the motor.

ABSPOS=*expression* sets the absolute position in units.

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

ABSPOS is the commanded motor position and is only set when 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.

Example

```
ABSPOS=2      'set absolute position to 2 units.
a=ABSPOS      'return ABSPOS value to variable "a".
ABSPOS        'return the current absolute position
```

ACCEL

Trajectory Parameter

Program Syntax **ACCEL=*expression***
ACCEL - used in an expression

Host Syntax **ACCEL=*number* cr**
ACCEL cr

Abbreviation **AC**

Description Sets the acceleration rate in units/sec^2. Specifies the rate at which the motor speed is increased. Specifying a 0 or negative value results in error code 6. Specifying a value greater than **Max Accel,** set in the system *Configuration and Setup,* results 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 is not 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.

Host Description Sets or returns the acceleration value of an axis. Setting ACCEL less than or equal to zero does *not* set Error Code 6.

Example

```
ACCEL=2      'Sets acceleration rate to 2 units/sec^2
a=ACCEL      returns the acceleration value to variable a
```

ANALOG

I/O Function

Program Syntax **ANALOG** - used in an expression

Host Syntax **ANALOG cr**

Abbreviation **AN**

Description Evaluates and returns the present analog input voltage. This value may vary for successive reads, but remains within the accuracy listed in the Hardware specification section of this manual.

Example

```
x=ANALOG     'sets variable x to the analog input voltage
ANALOG       'returns the analog input voltage value
```

AND

Boolean Operator

Syntax ***expression1*** **AND** ***expression2***

Description The logical AND operator is used in boolean expressions. The AND operator uses the following 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.

Example 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 the INDEX label.

ASC

String Manipulation

Syntax **ASC(*n*$)**

Description Returns the ASCII code for the first character in the string variable, *n*$. If the string is a null string, 0 is returned.

Example

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

BCD

I/O Operator

Syntax **BCD** - used in an expression
Host Syntax **BCD cr**
Description Evaluates and returns the BCD switches as a signed Integer value. The BCD switches are restricted to seven digits with a sign.

BCD takes precedence over OUT and toggles OUT3-OUT6 when called to strobe the BCD switch bank. Subsequent to a BCD call, an OUT(3)-OUT(6) **also sets the output** to the appropriate state. If OUT3-OUT6 are used as general purpose outputs, care must be taken not to invoke BCD or the state of the outputs are disturbed.

Example

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

BOOST

Motion Parameter

Syntax **BOOST=expression**
BOOST - used in an expression

Description Enables or disables the Boost Current feature of a stepper or returns the commanded boost status. If the expression is true (non-zero), BOOST is enabled, the current is boosted to the project configuration boost percentage setting during motion. If the expression is false (zero), BOOST is disabled, the current is the normal current during motion.

BOOST used in an expression returns the current setting of BOOST.

Example

```
BOOST=1      ‘ enables BOOST during motion
BOOST=0      ‘ disables BOOST during motion
X=BOOST      ‘ returns the current setting for BOOST
```

BUSY

Motion Parameter

Syntax **BUSY** - used in an expression

Host Syntax **BUSY cr**

Abbreviation **BS**

Description Returns the motion status. If the commanded motion is incomplete, BUSY returns a true (1). Otherwise, BUSY returns a false (0).

Example

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

CHR$

String Manipulation

Syntax **CHR$(*code*)**

Description Returns a one character string whose ASCII code is the argument. CHR$ is commonly used to send a special character to the serial port.

Example

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

CURRENT

Miscellaneous

Host Syntax **CURRENT cr**

Description This command only applies to the SS2000D6i/D3i controllers. It returns the Drive Current setting in amperes of the user project.

Example

```
CURRENT                 ‘returns the drive current setting
```

DECEL

Trajectory Parameters

Program Syntax **DECEL=*expression***
DECEL - used in an expression

Host Syntax **DECEL=*number* cr**
DECEL cr

Abbreviation **DC**

Description Sets the deceleration rate value in units/sec^2. The rate at which the motor speed is decreased. Specifying a 0 or negative value results in error code 7. Specifying a value greater than **Max Accel** (set in the *Configuration and Setup*), results 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 is not 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.

From the Host, setting DECEL less than or equal to zero does not set Error Code 7.

Example
```
DECEL=3.1     'sets the deceleration value to 3.1 units/sec^2.
X = DECEL     'sets variable X equal to the deceleration value.
```

DEFINE

Miscellaneous Command

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

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

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

replacement text is any sequence of letters of characters.

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

Example

```
#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
```

DIR

Miscellaneous

Host Syntax **DIR cr**

Description Returns the user project information. If there is no user project, DIR returns a cr-lf. 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 are used to identify a project, the trailing characters are 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

Example

```
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 Parameter

Syntax **DIST = *expression***
DIST - used in an expression

Host Syntax **DIST=*number* cr**
DIST cr

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

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, the move stops. 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.

Example

```
x=DIST
'sets x to the distance moved from the start of motion.
MOVEI = -25
DIST = -10        ‘shortens the move by 10 units
DIST      'returns the distance traveled from the start of motion.
```

Do…Loop

Program Flow

Syntax 1	Syntax 2
DO {UNTIL | WHILE} [*condition*]	**DO**
[*statement block*]	**[*statement block*]**
[EXIT DO]	**[EXIT DO]**
[*statement block*]	**[*statement block*]**
LOOP	**LOOP {UNTIL | WHILE} [*condition*]**

Description Repeats a block of statements **while** a condition is true or **until** a condition becomes true. 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 always executes 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 only be used in a DO...LOOP statement.

Example

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

ENCPOS

Trajectory Parameter

Syntax **ENCPOS** - used in an expression
ENCPOS

Host Syntax **ENCPOS cr**

Abbreviation **EP**

Description 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. If the unit is configured as an open loop stepper, ENCPOS is invailid.

Example

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

ENCSPD

Trajectory Parameter

Syntax **ENCSPD** - used in an expression

Host Syntax **ENCSPD cr**

Abbreviation **ES**

Description Returns the current encoder speed in units/sec. ENCSPD evaluates and returns the current encoder speed. 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*. For 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.
ENCSPD ' returns the current encoder speed.

END

Program Flow Control

Syntax **END**

Description Signifies the end of a program and must be included in each program or an error condition occurs.

Example *statement*
....
END

ERR

Return Error Code

Syntax **ERR** - used in an expression

Host Syntax **ERR cr**

Description Returns the error status of the controller. If an error occurs while the program is running, it jumps to the ERROR_ HANDLER label (if present). Otherwise, it ends. The fault LED is on while the error code is non-zero. ERR or GOTO 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.

Error Code	Description
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 an error handler routine is not present in the program, an error simply terminate program execution. Otherwise, an error causes the program to jump to the error handler routine (label **ERROR_HANDLER**). The error handler routine cannot be interrupted. The error handler routine is terminated with either END or GOTO <*label*> .

END terminates program execution. GOTO <*label*> causes program execution to continue with the line, <*label*>. At this point, the program can be interrupted.

Example

```
x=ERR
'Sets x equal to the present controller error number for this task and clears the error number.
ERR cr                    ' return the error status.
```

ERRM

Return Error Code

Host Syntax **ERRM cr**

Description Returns the error status of the controller.

Example

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

Event1

Motion

Syntax **EVENT1=*expression***

Host Syntax **EVENT1=*number* cr**

Abbreviation **E1**

Description Sets the trigger polarity and enable for MOVEHOME and MOVEREG. EVENT1 sets the EVENT1/IN1 input. This input is typically wired to a switch or sensor and is used as a home position trigger during a MOVEHOME cycle or a position mark registration trigger during a MOVEREG cycle. For mark registration, a trigger on EVENT1 initiates the index portion of the MOVEREG cycle. EVENT1 triggering for a MOVEHOME or MOVEREG cycle may be combined with an encoder index pulse input, and is assigned in *Configuration and Setup*.

For a **MOVEHOME** cycle, EVENT1 sets the polarity of the move home trigger. If the expression to the right of EVENT1 is **positive** (EVENT1 = 1), the home cycle trigger occurs when the EVENT1 input becomes active. If the expression to the right of EVENT1 is **negative** (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, EVENT1 sets the polarity of the registration trigger. If the expression to the right of EVENT1 is **positive** (EVENT1 = 1), the registration cycle trigger occurs when the EVENT1 input becomes active. If the expression to the right of EVENT1 is **negative** (EVENT1 = -1), the registration cycle trigger occurs when the EVENT1 input becomes inactive. The EVENT1 trigger for a registration cycle is **disabled** using EVENT1=0. A registration trigger is enabled to either polarity during a move. It cannot be disabled once the cycle has begun. EVENT 1 input state is read with IN(1).

Example

```
EVENT1=0
'disables trigger if assigned as a MOVEREG trigger.
EVENT1=1
'sets trigger to positive polarity triggering and enables the trigger.
EVENT1=-1
'sets trigger to negative edge triggering and enables the trigger.
```

Event2

Motion

Syntax **EVENT2=*expression***

Host Syntax **EVENT2=*number* cr**

Abbreviation **E2**

Description Sets the trigger polarity and enable used in MOVEHOME and MOVEREG. EVENT2 selects the effect of the hardware signal at the EVENT2/IN2 input (typically wired to a switch or sensor). It is used as a home position trigger during a MOVEHOME cycle or a position mark registration trigger during a MOVEREG cycle. When used for mark registration, a trigger on EVENT2 initiates the index portion of the MOVEREG cycle. Triggering is assigned in *Configuration and Setup*.

For a **MOVEHOME** cycle, EVENT2 sets the polarity of the move home trigger. If the expression is **positive** (EVENT2=1), the home cycle trigger occurs when EVENT2 becomes active. If the expression is **negative** (EVENT2= -1), the home cycle trigger occurs when EVENT2 becomes inactive. An EVENT2 home trigger cannot be disabled using this command.

For a **MOVEREG** cycle, EVENT2 sets the polarity of the registration trigger. If the expression is **positive** (EVENT2=1), the registration cycle trigger occurs when EVENT2 becomes active. If the expression is **negative** (EVENT2= -1), the registration cycle trigger occurs when EVENT2 becomes inactive.

The EVENT2 trigger for a registration cycle is disabled by EVENT2=0. A registration trigger is enabled to either polarity during a move. It may not be disabled once the cycle has begun. Read the EVENT 2 input state with IN(2).

Example

```
EVENT2=0
'disables trigger if assigned as a MOVEREG trigger.
EVENT2=1
'sets trigger to positive edge triggering and enables the trigger.
EVENT2=-1
'sets trigger to negative edge triggering and enables the trigger.
```

FEEDHOLD

Motion

Host Syntax **FEEDHOLD cr**

Abbreviation **FH**

Description FEEDHOLD stops all motion by decelerating at the programmed decel rate. The program continues to run. To continue motion, issue a RUN host command or toggle the RUN hardware input from inactive to active. To cancel the motion, issue a **<Ctrl-A>** host command.

Example FEEDHOLD cr

FOLERR

Trajectory Parameter

Syntax **FOLERR=*expression***
FOLERR - used in an expression

Host Syntax **FOLERR=*number* cr**
FOLERR cr

Abbreviation **FE**

Description 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 difference between the commanded and actual motor position (|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 FOLERR, the action taken depends 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 all motion is terminated. If error action is set to **correct on error**, the motor attempts to correct the error by making a new move to the required position. This move is 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, FOLERR is set to the maximum value (32767). Reading FOLERR returns the present setting in user units.

From the Host, 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. Entering a negative value does not set error code 19.

Example

```
FOLERR=.5       ' position error limit is set to 0.5 units
x=FOLERR        ' returns the current position error limit.
```

For Next

Program Flow Control

Syntax

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

Description

Repeats a block of statements a specified number of times.
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*, the loop will not execute, control is transferred to the statement following the NEXT statement. If *start* equals *end*, the loop executes once.

EXIT FOR exits 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 only be used in a FOR...NEXT statement.

Example

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

FREEMEM

Miscellaneous

Host Syntax **FREEMEM cr**

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

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, do not save the source code.

Example

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

GETCHAR

String Manipulation

Syntax **GETCHAR(*n*)** - used in an expression

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

Example

```
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

Program Flow Command

Syntax **GOSUB [*linelabel*]**

Description Branches to, and returns from, a subroutine. 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.

RETURN 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.

Example

```
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 subroutine
:
RETURN      'return to program line after GOSUB GET_CHAR
```

GOTO

Program Flow

Syntax **GOTO [*label*]**

Description Branches unconditionally to the specified label. 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 is difficult to read and debug. **Try to avoid using "GOTO"!**

Example

```
If x=1 Then GOTO coolant_off
:
:
coolant_off:
(statements)
```

HARDLIMOFF

Over-Travel Limit

Syntax **HARDLIMOFF**

Host Syntax **HARDLIMOFF cr**

Abbreviation **HL0**

Description Disables the hardware limit inputs. Inputs 3 and 4 are general purpose inputs with this command.

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

HARDLIMON

Over-Travel Limit

Syntax **HARDLIMON**

Host Syntax **HARDLIMON cr**

Abbreviation **HL1**

Description Enables the hardware limit inputs. Hard limit inputs stop the motor before it runs into a physical end of travel and avoid 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 are the +Limit and -Limit inputs. Configure the active signal level hard limits in the project's *Configuration and Setup* as either active on switch closing or active on switch opening.

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. Read the state of the +Limit with IN(3) and the state of the -Limit with IN(4).

Example
```
HARDLIMON       ' Hard limit inputs are active
```

HEX$

String Manipulation

Syntax **A$=HEX$(*expression*)**

Description Returns the hex string of an integer value. (*expression*) must be an integer value.

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

HVAL

String Manipulation

Syntax ***x*=HVAL(A$)**

Description Returns the decimal value of a hexadecimal string. A$ is the designated string variable or string literal. The converted value is an integer. *x* must be defined as an integer.

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

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

Program Flow Control

Syntax 1 **IF *condition* THEN *thenpart* [ELSE *elsepart*]**

Syntax 2
IF *condition1* THEN
[*statement block1*]
[ELSE] 'ELSE and statement block2 is optional
[*statement block2*]
END IF

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

condition is an expression that MC-BASIC evaluates as true (nonzero) or false (zero).

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

[statement block] includes any number of statements on one or more lines.

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.

Example
```
If x=0 Then
    statement block
Else
    statement block
End If
```

IN

I/O Operator

Syntax **IN(*nn*)** - used in an expression

Host Syntax **IN(*nn*) cr**

Abbreviation **I**

Description Returns the state of a digital input. *nn* is the specified digital input 1-17. The value returned is 1 for active or 0 for inactive. The inputs are assigned as:

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.

IN(*X*) returns 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, IN(7) returns a 1.

In the Host, setting *nn* greater than 17 does not set Error Code 9.

Example

```
IF IN(6)=1 Then GOTO continue
IN(6)     ' From Host, current state of input 6 is returned.
```

INCHAR

String Manipulation

Syntax **INCHAR(*n*)**

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

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

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

Example

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

INCLUDE

Miscellaneous Command

Syntax **#INCLUDE *drive*:*subdir*\...*subdir**filename*.inc**

Description Includes a file name with define statements in a user task. *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 only a series of #DEFINE statements 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.

Example

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

INPUT

String Manipulation

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*

Description

Reads a Line of data from the designated serial port into a string variable. 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 comma. The input arguments can be strings, integer values and real values. INPUT#1 designates the Host port and INPUT#2 designates the User port as the serial receiver port.

Example

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

String Manipulation

Syntax

INSTR(*string1*$,*string2*$) - used in an expression

Description

Returns the character position of the first occupance of a specified string in another string. The expression must be an integer variable. The comparison is case sensitive and returns a 0 if no match is found.

Example

```
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

Syntax **INTROFF*n***

Description Disables an interrupt for an ON...INTR*n* command. An interrupt causes the program to stop what it is doing, go do something else and 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 individually enabled or disabled. 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. *n* (1-4) defines the interrupt number to be disabled.

Example INTROFF1 'disables interrupt 1 for ON...INTR*n*.

INTRONn

Interrupt

Syntax **INTRON*n***

Description Enables an interrupt for an ON...INTR*n* command. An interrupt causes the program to stop what it is doing, go do something else and 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 individually enabled or disabled. 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.

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

Example INTRON1 'enables interrupt 1 for ON...INTR*n*.

JOG

Motion

Syntax **JOG = *expression***

Host Syntax **JOG = *number* cr**

Abbreviation **J**

Description Jog the motor in a specified direction. The sign of *expression* determines the direction of motion. If *expression* is positive or 0, jogging takes place in the positive direction.

If *expression* is negative, jogging takes place in the negative direction. The jog speed is determined by the last SPEED command.

Use STOP to stop the motor.

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

LCASE$

String Manipulation

Syntax ***string1*$=LCASE$(*string2*$)**

Description Converts and returns a string with lower case letters. **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.

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

LEFT$

String Manipulation

Syntax ***string2*$=LEFT$(*string1*$,*n*)**

Description Returns the leftmost characters of a string. *n* is the number of leftmost characters to return. If *n* is greater than the length of *string1*$, the entire string is returned to *string2*$.

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

LEN

String Manipulation

Syntax **LEN(*string*$)** - used in an expression

Description Return the number of characters in the designated string. The expression should be an integer type. A null input string returns 0.

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

LOWSPD

Trajectory Parameter

Syntax **LOWSPD=*expression***
LOWSPD - used in an expression

Host Syntax **LOWSPD=*number* cr**
LOWSPD cr

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

LOWSPD=*expression* sets the starting speed in units/sec. Set LOWSPD to either a positive number or 0. The default is 0.

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

Example
```
LOWSPD=2.5     ‘sets the starting speed to 2.5 units/sec
X=LOWSPD       ‘sets X equal to the present starting speed
LOWSPD         'returns the current starting speed
```

MID$

String Manipulation

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

Description Returns the designated middle number of characters of a string.

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

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.

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

MOVEA

Motion

Syntax **MOVEA=*expression***

Host Syntax **MOVEA = *number* cr**

Abbreviation **MA**

Description Moves the motor to the specified absolute position (less than ±2,147,483,647 microsteps away or error code 21 is set and no motion occurs). *expression* is the specified absolute position.

From the Host, entering a move distance more than 2,147,483,647 counts does not set error code 21.

Example
```
MOVEA= -1.0     ' moves to an absolute position of  -1.0 units.
MOVEA=2.5       ' moves to absolute position of 2.5 units.
```

Move Home

Motion

Syntax **MOVEHOME=*expression***

Host Syntax **MOVEHOME = *number* cr**

Abbreviation **MH**

Description *expression* determines the direction (positive or negative) of motion for the home cycle. The trigger is the EVENT 1 input, EVENT 2 input, or an Encoder marker state and is defined in the *Configuration and Setup*, and EVENT1 or EVENT2 (if executed prior to MOVEHOME).

Prior to starting a MOVEHOME, the appropriate trigger input is checked to see if it is triggered. If triggered, ABSPOS and ENCPOS are zero and no motion occurs. If not triggered, the motor accelerates 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. Once stopped, the distance traveled from the trigger becomes the new ABSPOS and ENCPOS value. The exact position the motor when the trigger occurred becomes the new **zero position**, or home.

Example
```
MOVEHOME= -1.0
'Initiates a mechanical home cycle in the negative direction.
MOVEHOME=0
'Moves motor back to electrical home. (i.e., switch edge)
MOVEHOME=+1
'Returns to mechanical home in the Positive direction
```

MOVEI

Motion

Syntax **MOVEI=*expression***

Host Syntax **MOVEI = *number* cr**

Abbreviation **MI**

Description Initiates an incremental move to *expression* distance from its present location. The sign of *expression* determines the direction (positive or negative) of motion for the move. The increment must be less than ± 2,147,483,647 microsteps or error code 21 is set and no motion occurs.

Example

```
MOVEI= -1.0          ' moves  -1.0 units.
MOVEI=2.5            '  moves +2.5 units.
```

MOVEREG

Motion

Syntax **MOVEREG=*expression***

Host Syntax **MOVEREG =*number* cr**

Abbreviation **MR**

Description Runs the motor until the mark registration input is activated Then moves the motor the desired registration distance. *expression* is the incremental distance to move after a registration trigger occurs. The sign of *expression* determines the direction (positive or negative) for the registration cycle. The distance must be less than ± 2,147,488,647 encoder counts or error code 21 is set and no motion occurs.

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

The Registration Travel Limit limits the distance the motor rotates if no trigger occurs. A **REGLIMIT** 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 **SPEED**.

When the registration trigger occurs, the registration distance is checked to determine if motion can be stopped in the given distance. If it cannot, motion is stopped using the project's *Configuration and Setup* setting for max. accel, and error code 134 is set. This error is eliminated by increasing the registration distance, decreasing the speed or increasing the deceleration.

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

Prior to starting MOVEREG, the appropriate trigger input is checked to see if it is itriggered. If triggered, an incremental move of the distance specified by the expression to the right of the MOVEREG occurs.

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

Example

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

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.
```

NOT

Boolean Operator

Syntax **NOT *expression***

Description The logical NOT operator is used in boolean expressions. NOT uses the truth table below: The result is TRUE if the expression is FALSE

Expression	Condition Result
True	False
False	True

Example

```
DO
:
:
LOOP UNTIL (NOT (BUSY))
‘The controller continues to execute until the axis completes the motion.
```

ON...INTRn

Interrupt

Syntax **ON [*condition*] INTR*n***

Description Sets condition to execute subroutine INTRn. *n* specifies the interrupt number 1-4.

When the specified condition for ON...INTR*n* becomes TRUE during program execution and the designated interrupt *n* is enabled, a subroutine call to label INTR*n* takes place. Upon completion of the subroutine, the program continues from where it was interrupted and executes the next program line. INTR*n* can be disabled at any time during program execution by using INTROFF*n*. INTR*n* is enabled at any time during program execution using INTRON*n*.

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

The following example shows the execution flow for two conditions to be tested. The first condition (TRUE) results 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 TRUE, jump to code at INTR1: If FALSE, continue with next program statement
PROGRAM STATEMENT	‘execute normal program line
program checks <*condition 1*>	'check condition 1, if TRUE, jump to code at INTR1: If FALSE, continue with next program statement

INTRON2	‘turn on interrupt2
program checks <*condition 1*>	'check condition 1, if TRUE, jump to code at INTR1: If FALSE, continue.
program checks <*condition 2*>	'check condition 2, if TRUE, jump to INTR2: If FALSE, execute next program statement.
NEXT PROGRAM STATEMENT	'execute next line in program
program checks <*condition 1*>	'check condition 1, if TRUE, jump to code at INTR1: If FALSE, continue.
program checks <*condition 2*>	'check condition 2, if TRUE, jump to INTR2: If FALSE, execute next program line.
INTROFF1	
program checks <*condition 2*>	'check condition 2, if TRUE, jump to INTR2: If FALSE, execute next program line.
INTROFF2	'condition 2 is not checked since Interrupt 2 is 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:	‘beginning of interrupt 1 routine
PROGRAM STATEMENTS	'execute program statement interrupt conditions are not checked after program statements within the interrupt routine.
RETURN	'end of interrupt 1 routine

```
INTR2:                          'beginning of  interrupt 2 routine
  PROGRAM STATEMENTS            '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. RETURN is required at the end of each subroutine. There are four interrupt subroutines labeled INTR1-INTR4.

Example

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

Syntax ***expression1*** **OR** ***expression2***

Description The OR operator uses the truth Table below. 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

Example

```
DO
LOOP until (A > 5 OR X = 0)
'The controller loops until variable A>5 or variable X=0.
```

OUT

I/O Operator

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

Host Syntax **OUT(*n*) = *number* cr**
OUT(*n*) cr

Abbreviation **O**

Description Sets/returns the condition of a digital output (*n* =1-8). OUT(*n*) returns 1 for a commanded active output and 0 for a commanded inactive output. OUT(*n*) = *expression* If *expression* is non-zero, the specified output is activated. From the Host, *n* greater than 8 does *not* set error code 8. Outputs:

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

Example

```
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.
```

BCD takes precedence over OUT and toggles OUT3-OUT6 when called to strobe the BCD switch bank. If OUT3-OUT6 are general purpose outputs, do not to invoke BCD or the output state is disturbed.

PRINT

String Manipulation

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

Description Transmits designated data via the designated serial port. 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 separate the expressions. If a semicolon is used between expressions, there is 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 is generated.

Example

```
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

String Manipulation

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*][;]

Description

Prints strings character or formatted numbers. Port 1 is the Host Port and Port 2 is the User Port.

The numeric values are formatted using only the literal string or a designated Format$ variable string. This string can contain non-format characters printed prior to the formatted number. The following characters in the string are not 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 comma 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, an asterisk is substituted for the 0s 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 is always 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 is always filled.
0	Represents a digit position. If no data exists at the digit position, substitutes a zero. The digit field is always filled.
.	A decimal point may be inserted at any position in the field.

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

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

The following example illustrates how the format string is processed.

Example:

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

"Numbers are " is extracted from the string and sent via serial port.

"+###.##" is extracted from the string as the data format, resulting in "+100.54" being sent via serial port.

The string, "mv," is sent via serial port.

" " is extracted from the string and sent via serial port.

"###" is extracted from the string as the data format, resulting in "999" being sent via serial port.

The string, "cnts," is sent via serial port.

" " is extracted from the string and sent via serial port.

"0##" is extracted from the string as the data format, resulting in "054" being sent via serial port.

The string, " is limit," is sent via serial port.

" **" 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>

Example

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

Syntax

REDUCE=*expression*
REDUCE - used in an expression

Description

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

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

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

Example

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 X.

REGLIMIT

Over-Travel Limit

Syntax **REGLIMIT** - used in an expression
REGLIMIT=*expression*

Host Syntax **REGLIMIT cr**
REGLIMIT=*number* cr

Abbreviation **RL**

Description 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.

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

REGLIMIT=expression sets the MOVEREG travel distance. Set REGLIMIT to either a positive number or 0. Setting REGLIMIT= 0 or a negative number allows MOVEREG to run indefinitely while awaiting a trigger. If REGLIMIT <0, REGLIMIT is set to 0.

From the Host, sets or returns the maximum mark registration distance before indicating an error and stopping motion.

Example
```
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

Host Syntax **RESET cr**

Description This command causes the system to halt, and then restart as though power has been cycled.

REVISION

Miscellaneous

Host Syntax **REVISION cr**

Abbreviation **REV**

Description Returns the current revision level of the controller's operating system software.The return format for this command is:

DCC REV *n.nn mm*/*dd*/yy

where:

n.nn	software revision
mm	month
d	day
yy	year

Example REV

RIGHT$

String Manipulation

Syntax ***string1*$=RIGHT$(*string2*$,*n*)**

Description Returns the rightmost characters of a string. *n* is the number of rightmost characters to return. If *n* is greater than the length of *string2*$, the entire string is returned to *string1*$.

Example

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

RUN

Miscellaneous

Host Syntax **RUN cr**

Description Starts the user program or resumes from a FEEDHOLD condition that has been generated either by the hardware input Feedhold or by the FEEDHOLD host command. Resumes motion if a Feedhold condition exists.

Example RUN

SOFTLIMNEG

Over-Travel Limit

Syntax **SOFTLIMNEG=*expression***
SOFTLIMNEG - used in an expression

Host Syntax **SOFTLIMNEG=*number* cr**
SOFTLIMNEG cr

Abbreviation **SLN**

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

Software travel limits stop the motor when 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 SOFTLIMNEG.

When a travel limit is exceeded, the motor decelerates 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.

Example

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

SOFTLIMOFF

Over-Travel Limit

Syntax **SOFTLIMOFF**

Host Syntax **SOFTLIMOFF cr**

Abbreviation **SL0**

Description Disables the software over-travel limits. This command disables the negative and positive software limits checking during motion.

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

SOFTLIMON

Over-Travel Limit

Syntax **SOFTLIMON**

Host Syntax **SOFTLIMON cr**

Abbreviation **SL1**

Description Enables the software over travel limits. Software travel limits stop the motor when 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.

SOFTLIMON 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 occurrs.

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

When a travel limit is exceeded, the motor is decelerates 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.

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

SOFTLIMPOS

Over-Travel Limit

Syntax **SOFTLIMPOS=*expression***
SOFTLIMPOS - used in an expression

Host Syntax **SOFTLIMPOS=*number* cr**
SOFTLIMPOS cr

Abbreviation **SLP**

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

Software travel limits stop the motor when 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 SOFTLIMPOS.

When a travel limit is exceeded, the motor decelerates 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 software travel distance.

Example
```
SOFTLIMPOS = +4
' Sets the absolute software travel distance to +4 units.
SOFTLIMPOS    ' returns the current SOFTLIMPOS value.
```

SPEED

Trajectory Parameter

Syntax **SPEED = *expression***
SPEED - used in an expression

Host Syntax **SPEED = *number* cr**
SPEED cr

Abbreviation **SPD**

Description Sets and returns the target velocity of the motor.

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 Configuration and Setup, results 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 and is effective immediately.

The lowest programmable speed is 0.00015 rev./second.

Example

```
SPEED=3.0      ' sets the velocity to 3.0 units/second.
x=SPEED        ' sets x to 3.0.
SPEED          ' returns 2.0
```

STOP

Motion

Syntax **STOP**

Host Syntax **STOP cr**

Abbreviation **S**

Description Stops motor motion using the programmed decel and velocity profile.

Although the motion in progress stops, the user program may continue to execute. If subsequent program statements call for motion, these new motion commands execute and motion occurs.

Example

```
STOP           ' generates a motion stop command.
               ' the present value of DECEL is used
               ' as the deceleration rate
```

STOPERR

Motion

Syntax **STOPERR=*expression***
STOPERR - used in an expression

Host Syntax **STOPERR=*number* cr**
STOPERR cr

Description Sets or returns the maximum position error allowed when motion is stopped (position error band).

STOPERR=*expression* specifies the maximum position error allowed when motion is stopped. Setting the position error band equal to zero disables 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, STOPERR is set to 0.005 revolutions.

STOPERR - used in an expression returns the current STOPERR value in units.

Example

```
STOPERR=.1      ‘sets the position error band to 0.1 units.
X=STOPERR       ‘returns the current STOPERR to variable X
STOPERR         ‘returns the current STOPERR.
```

STR$

String Manipulation

Syntax ***string1*$=STR$(*numeric_expression*)**

Description Returns a string representation of a numeric expression. The numeric expression can be a parameter value, real value or integer value.

STR$ is the complement of VAL.

Example

```
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$

String Manipulation

Syntax ***string1*$=STRING$(*num*,*code*)**

Description Returns a string of characters. *num* is the length of the returned string. *code* is the ASCII code of each character.

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

TIMER

Time Function

Syntax **TIMER=*expression***
TIMER - used in an expression

Description Sets or returns the timer value.

TIMER = *expression* sets the timer value (in seconds) to the expression.
TIMER returns the current timer value to the variable.

The timer is free running and counts **up** in 0.001 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 toward zero (i.e., the next count is -2,147,481.646). Programs using large timer values must take this into account and adjust appropriately.

Example
```
TIMER=0                      'sets the Timer value to 0.
DO
  statements
LOOP WHILE TIMER < 1.0   'Do this loop until timer >= 1.0
```

UCASE$

String Manipulation

Syntax ***string1*$=UCASE$(*string2*$)**

Description Converts and returns a string with upper case letters. *string2*$ is copied. All lower case letters are converted to upper case letters. The resulting string is returned *string1*$. This command is useful for making INSTR case insensitive.

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

UNITID

Daisy Chain

Syntax **UNITID** - used in an expression

Description Returns the current unit ID value (1-32) read from the unit ID switches on power-on.

Example
```
ID =UNITID          ‘ sets variable ID to the unit ID number
IF VAL(unitid$) = ID Then
‘ if received ID matches the unit ID, execute statement block
    [statement block]
END IF
```

VAL

String Manipulation

Syntax **VAL(*n*$)** - used in an expression

Description Return the numeric value of string *n*$. 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.

Example
```
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 Function

Syntax **WAIT=*expression***

Description Waits for the specified period of time (seconds) to expire before continuing.

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

WAITDONE

Time Function

Syntax **WAITDONE=*expression***

Description Waits for motion to be complete before program execution continues. An alternate way to accomplish this is:

```
DO
:
LOOP WHILE BUSY       ' Waits until motion is completed.
```

Example

```
WAITDONE
‘Waits for motion to complete before continuing.
```

WNDGS

Motion

Syntax **WNDGS=*expression***
WNDGS - used in an expression

Host Syntax **WNDGS=*number* cr**

Abbreviation **WN**

Description Enables/disables the stepper drive motor current. The stepper drive powers up with the motor current flowing (enabled) at standstill. WNDGS controls the drive enable output at standstill. WNDGS=1 enables the stepping motor, winding current is on when there is no motion.WNDGS=0, winding current is off when there is no motion.

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

If the stepper motor is configured as closed loop, position maintenance is disabled when WNDGS = zero. The encoder position is maintained while the drive is disabled. A position error can occur when WNDGS is re-enabled.

WNDGS=*expression* sets the no motion drive current state. *expression* must be either zero or a positive number.

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

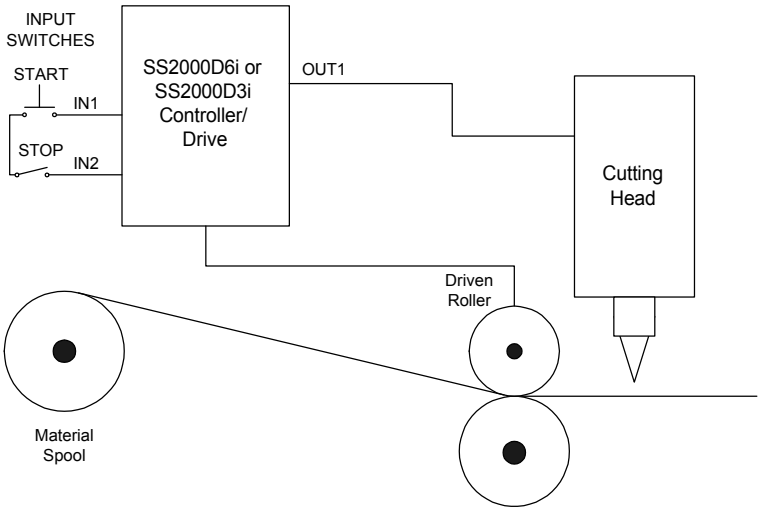
Example

```
WNDGS=1   ‘stepper drive current is normal with no motion
WNDGS=0   ‘ stepper drive current is off with no motion
X=WNDGS   ‘ returns the current WNDGS state to X.
```

5. PROGRAMMING EXAMPLES

This section contains three sample applications easily implemented using the SS2000D6i Controller/Drive. Each contains an explanation of the application, including requirements and a diagram. A sample program, including comments is also provided for each application.

5.1. Cut to Length Application



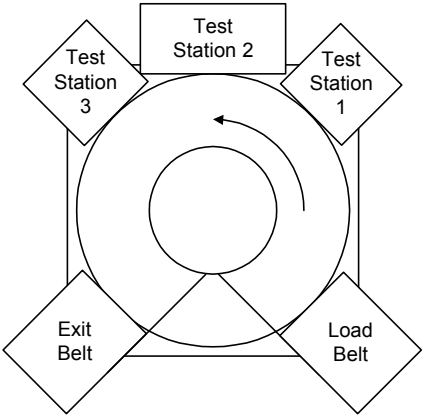
This application requires a stepper motor to run a pair of nip rollers that draw material from a spool. The material could be anything from paper to steel. The requirements for this application are:

- 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, 0.25 sec.
      if in(2)=0 then goto begin 'Return to beginning of program.
end                             'End of program.
```

5.2. Rotary Table Application with Test Stations



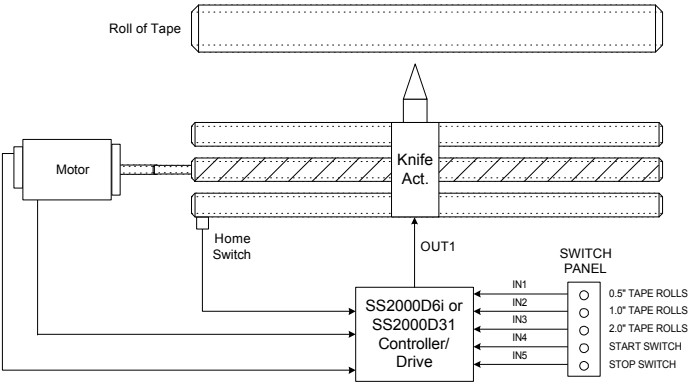
In this application, a part is loaded onto a rotary table via a load belt. Once the part is on the table, it is tested at three test stations. The application requires:

- A sensor to tell the table to jog until a sensor indicates the presence of a part on the table.
- Outputs to tell the load and exit belts to stop until testing is complete at all the test stations.
- Rotate the table 90° to position the part at the first test station.
- The sample to be at each test station until an input is activated on the control. Each test station is 45° apart.
- The part to rotate 90° after the last test station to the exit station where it is then carried out via the exit belt. A sensor tells the controller to start the load belt for the next part.

Rotary Table Program Code

```
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 90°
    waitdone                           ' Wait for motion to be completed
    do: loop until (in(2)=1)           ' Wait for test station to test and turn on input 2.
    count =0                           ' Initialize a counter to zero
    do                                 ' Do loop begin
movei=+45                              ' Move rotary table +45° to the next station
waitdone                               ' Wait until the motor stops.
do: loop  until(in(3)=1 or in(4)=0)    ' Loop until test stations 2 and 3 complete.
count=count+1                          ' Increment counter by 1.
    loop until count=2                 ' Do loop end.
    movei=+90                          ' Move the part 90° to the exit belt
    waitdone                           ' Wait for motion to be completed
    if in(5)=0 then                    ' Check input 5. If inactive, 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.
```

5.3. 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 is determined by the switch inputs from a switch selector panel. The machine operates 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.

Rotary Table Program Code

```
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.
```

6. TROUBLESHOOTING

The following table lists some of the more commonly encountered problems and possible solutions.

Problem	Possible Solutions
Cannot establish communication.	1) Insure that the connections are correct. Review serial port communication connections. 2) Check to make sure that the communication parameters are set correctly. 3) Check to see if there is power to the unit.
When I press a button connected to my run input, the motor does not turn.	1) The switch may not be connected properly.Review Input/Output Connections for diagrams on how to properly wire the run input (IN 5). 2) The motor could be connected improperly. Refer to Motor and Encoder Connections, to ensure the motor connectrions are correct. 3) Check to see if there is power to the unit. 4) The program is 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. 2) Verify that your machine is not mechanically bound-up or obstructed.

Problem	Possible Solutions
When my system activates a sensor, the controller does not recognize it.	1) Check to make sure you are operating in the correct mode. a. An NPN sensor requires sink mode. b. A PNP sensor requires source mode. 2) Check the connections and make sure you have power to the sensor.

If more information is needed or additional assistance is required, contact Customer Support.

7. GLOSSSARY

A

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 that keeps track of the commanded motor position. When the value in this register is zero, the position is electrical home.

ACCELERATION
The rate at which the motor speed is increased from its present speed to a higher speed (specified in units/second^2).

ACCURACY (of step motor)
The noncumulative incremental error that 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
(**A**merican **S**tandard **C**ode for **I**nformation **I**nterchange). A format to represent alphanumeric and control characters as seven-or eight-bit codes for data communications. A table of ASCII codes is on the following page.

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	"	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

ATTENTION CHARACTER

<*nn*, where *nn* is a unique integer from 1-99 (set by use of the unit ID# select switches) assigned to a motion controller arrayed in a multi-controller system. The attention character directs the program command to the specified motion controller.

B

BACK EMF
The voltage a permanent magnet 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 (KE) 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 - (**B**inary **C**oded **D**ecimal)
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 required to bring the motor from a running condition to a stop. This also describes the torque developed during a dynamic braking cycle.

C

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.

D

DAISY-CHAIN
A method to interface multiple motion controllers via RS-485 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^2).

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 on-time versus off-time. This is usually expressed in terms of a percentage of on-time versus the total time given by the following equation:

$$D_{Cycle} = \frac{T_{ON}}{T_{OFF} + T_{ON}} \times 100$$

DWELL
See WAIT.

E

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 is used to determine position, speed or acceleration. The encoder may be either absolute or incremental.

F

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.

H

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 with another's. The motion controller uses a software XON, XOFF handshake method. See XON.

HOST
The computer or terminal connected to the HOST serial port on the motion controller and is responsible for primary programming and operation of the controller.

I

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.

J

JOG MOVE
Moves the motor continuously in a specified direction.

L

LOAD
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 computer to another. DOWNLOAD is transmitting a program from a host computer (where a program has been written) to the motion controller where it is used. UPLOAD is transmitting a program from a motion controller back to the host computer.

M

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 that allows the motion controller to move the motor and seek a switch to establish electrical home and set Absolute Position = zero.

N

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.

O

OPTO-ISOLATION
The electrical separation of the logic section from the input/output section to achieve signal separation and limit electrical noise. The two systems are coupled together via a transmission of light energy from a sender (LED) to a receiver (photo transistor).

P

PARITY
An error-checking scheme used in serial communications (via the RS-232 or RS-485 port) to ensure that the data received by a motion controller is the same as the data sent by a host computer or terminal.

R

REGENERATION
A condition when the motor enters a braking-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. ***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 that produces 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 that produces 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.

S

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 accessed from anywhere in a program to preclude having to repetitively program those lines. This allows shorter, more powerful, and more efficient programs. See also NESTING.

T

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 (translator drive) that converts input pulses to motor phase currents to produce motion.

TTL
Also called T^2L, **T**ransistor - **T**ransistor - **L**ogic

V

VOLTAGE CONSTANT (or BACK EMF CONSTANT)
A number representing the relationship between the back EMF voltage and angular velocity. Typically expressed in:

$$\frac{\text{volts}}{\text{1000 rpm}}$$

W

WAIT
A programmed delay or dwell in program execution (specified in seconds).

X

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 are 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 resume when the controller receives an XON character.

APPENDIX A

A.1 CE Compliance Installation Requirements and Information

Certain practices must be followed when installing the WARPDRIVE™ SS2000D6i or SS2000D3i controller/drive 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. You must ensure the system or end product complies with all applicable standards required for the equipment, including overall CE certification. Following these practices will help ensure (but cannot guarantee) that the machine in which these components are utilized meet overall CE requirements.

A.1.1 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, the SS2000D6i controller/drive meets all the applicable EMC emission and immunity standards on a stand-alone basis:

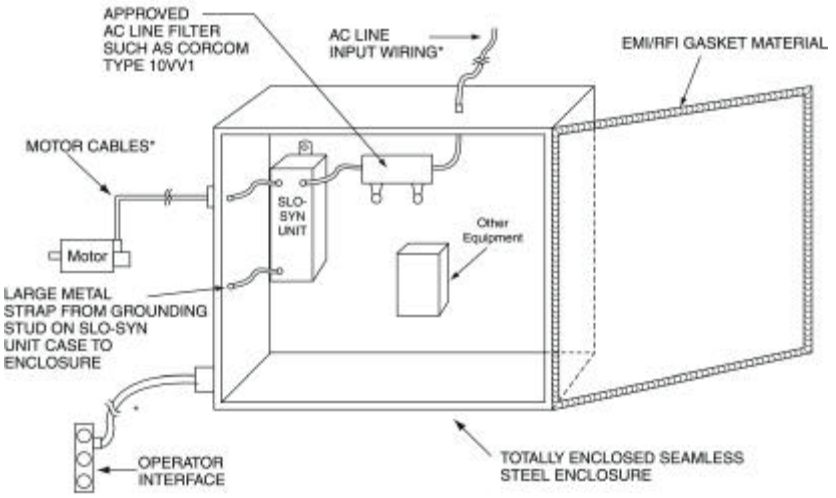
EN55011, Class A:	for Radiated and Conducted Emissions
IEC1000-4-3:	for RF Radiated Immunity (RFRI)
IEC1000-4-4:	for Electrical Fast Transient Immunity (EFT)
IEC1000-4-6:	for RF Conducted Immunity (RFCI)

In order to achieve full CE compliance, an additional requirement must be met:

IEC1000-4-2:	for ESD Immunity

To meet this requirement, the unit must be placed inside a metal enclosure, as shown in Figure 1.

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.

A.1.2 Low Voltage Directive

1) These drives are to be operated in a pollution degree 2 environment as described in standard EN50178.

2) All of the control inputs and outputs are isolated from the main input power with a basic insulation rating (their impulse withstand voltage capability is 2.5kV (1.2 / 50 µs) as referenced in EN50178). 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 your responsibility to provide this protection, if needed.

3) All cautions and warnings listed throughout the operators manual MUST be followed to insure safe system operation.

A.2 WARRANTY

Danaher Motion, warrants to the first end user purchaser (the "purchaser") of equipment manufactured by Danaher Motion 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 Danaher Motion's factory or a warehouse of Danaher Motion in the event that the equipment is purchased from Danaher Motion or for a period of one year from the date of shipment from the business establishment of an authorized distributor of Superior Electric in the event that the equipment is purchased from an authorized distributor.

DANAHER MOTION'S OBLIGATION UNDER THIS WARRANTY SHALL BE STRICTLY AND EXCLUSIVELY LIMITED TO REPAIRING OR REPLACING, AT THE FACTORY OR A SERVICE CENTER OF SUPERIOR ELECTRIC, ANY SUCH EQUIPMENT OF 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. DANAHER MOTION 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 Danaher Motion; and no person is authorized to assume for Danaher Motion any other liability with respect to equipment manufactured by Danaher Motion. Danaher Motion shall have no liability with respect to equipment not of its manufacture. DANAHER MOTION 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 Danaher Motion. Danaher Motion shall not be responsible for any transportation charges.

IF FOR ANY REASON, ANY OF THE FOREGOING PROVISIONS SHALL BE INEFFECTIVE, DANAHER MOTION'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 Danaher Motion 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.

The right to make engineering refinements on all products is reserved. Dimensions and other details are subject to change.

Artisan Technology Group is an independent supplier of quality pre-owned equipment

Gold-standard solutions

Extend the life of your critical industrial, commercial, and military systems with our superior service and support.

We buy equipment

Planning to upgrade your current equipment? Have surplus equipment taking up shelf space? We'll give it a new home.

Learn more!

Visit us at **artisantg.com** for more info on price quotes, drivers, technical specifications, manuals, and documentation.

Artisan Scientific Corporation dba Artisan Technology Group is not an affiliate, representative, or authorized distributor for any manufacturer listed herein.

We're here to make your life easier. How can we help you today?

(217) 352-9330 | sales@artisantg.com | artisantg.com

ARTISAN® TECHNOLOGY GROUP