PRICE: $25.00

INSTALLATION AND PROGRAMMING INSTRUCTIONS for the SLO-SYN® TDC SERIES SERVO DRIVE/CONTROL

Superior Electric

ENGINEERING CHANGES

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

Revision	Date	Description
A	6/3/97	Original Issue
B	11/20/97	Revise ISO Logo, Corporate ID, and Switch SW1 default positions.
C	4/3/98	Add Appendix A for CE compliance
D	10/01/98	Revise Sections 5 and 6 to reflect the MCPI sofware configuration and add Daisy chaining information.
E	04/22/99	Revise Appendix A, remove "pr" prefix from standard EN50178
F	08/06/99	Add information on NVR and Electronic Gearing (new section).
G	1/25/2000	Revise Corporate ID

Table of Contents

SECTIONS & TITLE **PAGE**

SECTIONS & TITLE **PAGE**

SECTIONS & TITLE **PAGE**

A

B

C

6.1.4 - Programming Commands - Alphabetical Listing (continued)

D

E

F

G

H

I

J

SECTIONS & TITLE **PAGE**

6.1.4 - Programming Commands - Alphabetical Listing (continued)

K

L

M

N

O

P

R

S

T

U

SECTIONS & TITLE **PAGE**

SECTIONS & TITLE **PAGE**

Section 1

Introduction

1.1 - HOW TO USE THIS MANUAL

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

1.2 - WHAT YOU NEED TO KNOW FIRST

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

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

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

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

1.3 - CONVENTIONS USED IN THIS MANUAL

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

2. Please refer to the AGlossary@ section for detailed descriptions of terms such as "sink and source I/O", various motion terms, etc.

1.4 - HOW TO CONTACT US

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

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

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

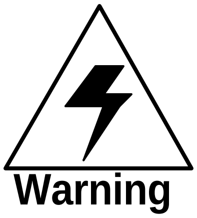
2. **By Phone**. You may reach us by phoning our Motion Control Applications Engineering Department at telephone (800) 787-3532 ext. 4751. Or call our main number at (860) 585-4510. Both may be reached between the hours of 8:00 am and 5:00 pm (Eastern Time), Monday through Friday. Technical personnel are available to assist you in getting your application up and running as efficiently as possible.

Section 2

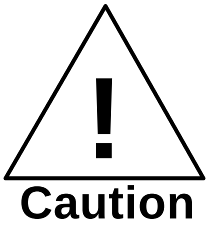
Important
Safety Information

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

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



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



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

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

Warning

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

Caution

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

Caution

Servo motors can have temperatures of up to or exceeding 100EC. Use caution when handling the motors.

Warning

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

Caution

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

⚠ Caution

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

⚠ Caution

External regenerative resistors can be a shock and temperature hazard. These resistors should be mounted and enclosed properly with safe clearances around them. Proper ventilation must also be provided for cooling purposes.

⚠ Caution

Secure mounting and proper grounding of both the Slo-Syn controller and the servo motor are essential for proper operation of the system.

⚠ Caution

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

⚠ Caution ⚡ Warning

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

⚠ Caution

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

used in these situations.

Section 3

Quick Start Installation Guide

3.1 - Step-by-Step Start-Up Procedure

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

Bench Set Up.

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

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

HINT: Don't forget to wire the User Enable signal to GND through a switch so you can turn the unit on and off as necessary.

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

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

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

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

5) Tune the Servo. Before running the motor, the controller compensation parameters (gains) must be set. To aid in this task an automatic servo tuning procedure is available. To enter the servo tuning screen click on the servo tuning button. The default values for auto tuning procedure should work fine for now. The motor may be tuned on the bench with no load. Ensure that the motor is properly secured to your work surface (bench). Note: Do not clamp the motor any where except at the mounting flange.

Begin the auto tuning process by clicking on the Auto Tune button. A screen with the default values will appear. Click OK to use these settings. Next, click the Measure System Gain button. The motor should Abump@ then the System Gain value should update on the screen. Now click on the Calculate Servo Gains button and the calculated Servo Gain value will be displayed on the screen. Click the Update Gains button, the servo should now be locked in position. Verify this by manually trying to turn the motor shaft. The servo should Afight@ to stay in position.

It's now time to try a test move by entering profile parameters. First click the Motion Setup button and enter the desired Accel, Decel, Speed and Move distance in user units (e.g. revolutions by default). When finished click Done. Now make the motor move by clicking on the Move response button. The motor should complete the programmed profile and the position error plot should appear on the screen. You may have to adjust the display time in order to see the whole move. To save the gain settings, click on the Quit button then the Yes to save. To complete the tuning Compile and Download the project to the controller to save these values. The servo gains are now set.

6) Make it move! Now that you have compiled and downloaded your project into the unit, and tuned the servo you are ready to command the motor to move. First you must enter the speed at which you wish the motor to turn, such as 1 rev/sec. Do this by typing speed=1 <CR> in terminal mode (the <CR> means the Return or Enter key). Now enter the acceleration, for example 50 revs/sec/sec by typing accel=50<CR>. Set the deceleration to match by typing decel=50<CR>. After each entry, the controller should respond with a ">" prompt indicating that it has accepted your command. In addition set wndgs to 1<CR> 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. Also verify your configuration settings. In addition, check the motor direction to insure it meets your requirements. The motor direction can be reversed in the System folder if necessary.

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

8) Execute the Program. From the Terminal screen, click on the RUN button to make the motor move 10 revolutions. If desired you can now add lines to the program to perform more sophisticated motion. For example, try typing REAL x <CR> as the first line of your program. This will declare x as a REAL variable. See sections 5 and 6 for discussions regarding variables. On the next line, type x=10 <CR>. This assigns the REAL variable x a value of 10. Change the movei=10 line to movei=x. Now the motor will move whatever distance has been assigned to x. Recompile and download your program, then run it. It should operate the same as before, but now the program is using x as the move distance in place of 10 as before. Change the value of x to different distance values to verify that it works correctly.

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

Installation into Mechanical System

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

3.1.1 - Switch and Jumper Settings

Caution

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

Warning

NEVER change the jumper or switch settings with the unit powered on. Risk of physical injury or damage to the unit. Before changing jumpers, place the unit on a properly grounded antistatic workstation to avoid static discharge damage to the board.

3.1.2 - 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 downloaded project. If the switch is in the 9600 position, the baud rate will be forced to 9600 regardless of the project configuration.

It is possible to communicate to multiple TDC units over the same RS-485 transmission lines. To accomplish this, the TDC supports daisy chain wiring of from 2 to 32 units. **All units MUST have their HOST communications port set to RS-485 mode for daisy chaining to function properly. Insure that the power is off when changing the setting.** To change the Host port communications mode remove the 8 screws holding the cover on and place the jumpers on JP4 & JP7 to the RS485 position, see Figure 3. Put the cover back on and secure the 8 screws holding the cover in place. All units must also be set to the same baud rate.

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

The host command <nn allows different modes for daisy chain communications. Refer to Section 6.2 for a detailed description of the daisy chain commands including syntax and usage.

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

The unit ID 's are decoded as follows:

ID Num.	SW-1	SW-2	SW-3	SW-4	SW-5
1	ON	ON	ON	ON	ON
2	ON	ON	ON	ON	OFF
3	ON	ON	ON	OFF	ON
4	ON	ON	ON	OFF	OFF
5	ON	ON	OFF	ON	ON
6	ON	ON	OFF	ON	OFF
7	ON	ON	OFF	OFF	ON
8	ON	ON	OFF	OFF	OFF
9	ON	OFF	ON	ON	ON
10	ON	OFF	ON	ON	OFF
11	ON	OFF	ON	OFF	ON
12	ON	OFF	ON	OFF	OFF
13	ON	OFF	OFF	ON	ON
14	ON	OFF	OFF	ON	ON
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

3.1.3 Controller Board Jumpers

There are five jumper banks which must be set on the controller board. **BEFORE** removing the unit cover, check to see if the factory default jumper settings are acceptable for your application. If changes must be made to the control board jumpers, first you MUST insure that the unit is NOT powered on. Next remove the sheet metal cover by unfastening the screws from the cover and carefully sliding it free of the unit. Lay the unit on its side to gain access to the jumpers. Carefully set the jumpers according to the Table 3.1 below.

TABLE 3.1

Jumper	Function	Setting 1 (Factory Default)	Setting 2
JP1	Mode: Output 2	Sink Mode 1-2, 3-4	Source Mode 2-3, 4-5
JP2	Mode: Output 1	Sink Mode 1-2, 3-4	Source Mode 2-3, 4-5
JP3	Mode: All OPTO Inputs	Sink (See Board Legend)	Source (See Board Legend)
JP4 & JP7*	Mode: Host Serial Port 1	RS232	RS485

*** NOTE: Both Jumpers JP4 and JP7 must be set to the same mode**.

3.1.4 Drive Current Switch

The drive current selection switch is accessible through the top center of the unit. The switches should normally be set to reflect the continuous current capability of the motor. The available peak current is twice the continuous current setting.

The TDC-04 has a maximum continuous current of 4 amps and a peak current of 8 amps. The TDC-08 has a maximum continuous current of 8 amps with 16 amps peak current. The **highest** DIP switch setting in the on position will be the maximum **CONTINUOUS** current supplied by the drive.

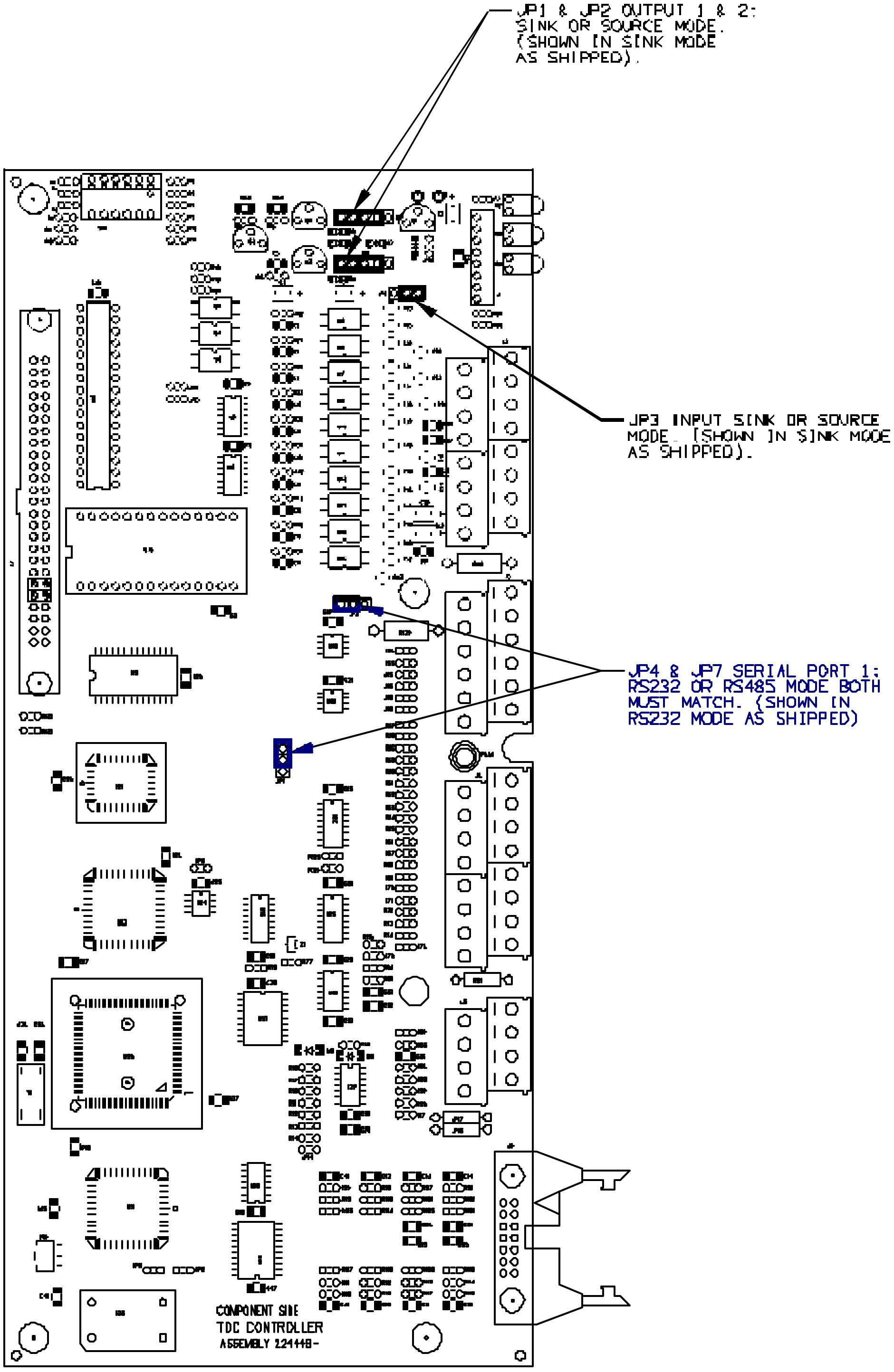


Figure 3.1
Control Board Jumpers

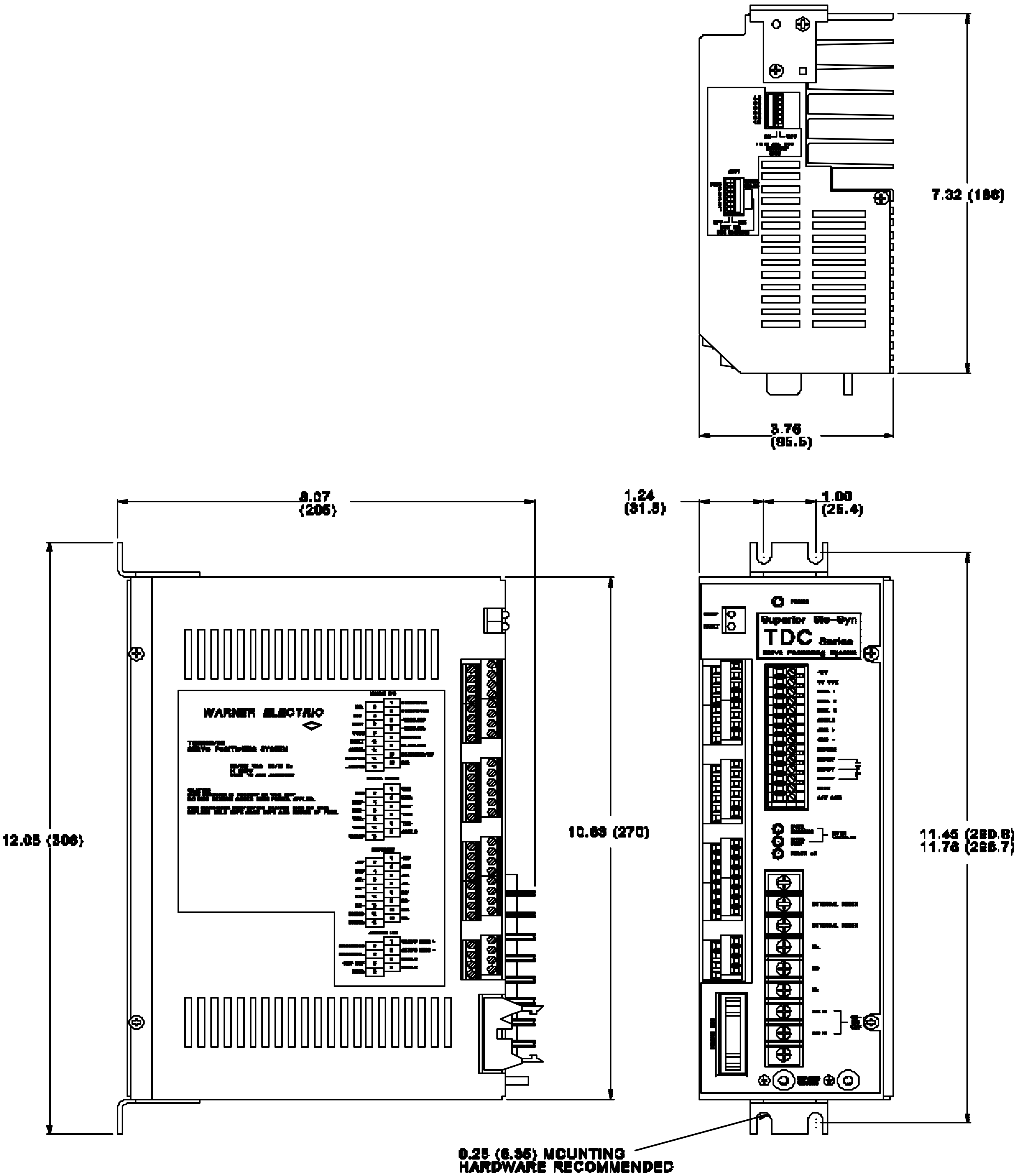


Figure 3.2
Mechanical Outline Drawing

3.2 Mechanically Mounting the Unit

Mechanical outline drawings are shown on the previous page. The unit should be solidly mounted within a control enclosure approved for the particular application. It is important to select a mounting location which will meet the environmental specifications listed in the Mechanical Specifications section of this manual. Avoid locations that expose the unit to extremes of temperature, humidity, dirt/dust, or vibration.
At least 2 inches of space must be left on the sides, top, and bottom of the unit to allow proper air flow for cooling of the unit.

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

3.3 General Wiring Guidelines

Warning

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

Warner 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 Apolluted@ with electrical noise from sources outside a user=s control (such as equipment in the factory next door). Some of the more common causes of electrical interference are:

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

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

Solid grounding of the system is essential. Be sure that there is a solid connection to the ac system protective earth ground (PE). Insure that there is a good electrical connection through the 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 0.5" in width. Use a single-point grounding system for all related components of the system (a Ahub and spokes@ arrangement). Keep the ground connection short and direct. Grounding through **both** a mechanical connection to the control enclosure and through a grounding strap is optimal.

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

Use separately bundled shielded, twisted-pair cables for the drive to motor, encoder, serial communications, analog input, and digital I/O wiring. For 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 ONE END ONLY. Grounding the serial communications connections at the opposite end of the controller may be necessary in some systems. If the cable shield must be connected at the opposite end from the Slo-Syn unit, it should NOT also be connected at the unit as this may cause a Aground loop@ and introduce electrical noise problems.

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

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

NOTE: Warner Electric makes a wide range of ac power line conditioners that can help solve electrical interference problems. Contact 1-800-SUP-ELEC for further assistance.

3.4 - Hardware Connection Descriptions

The following figures indicate the side, top, and front views of the TDC controller. The numbers in the boxes show the position of the various hardware connections to the unit. Use the index number in the boxes to find the description of each of the connections 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 of this manual.

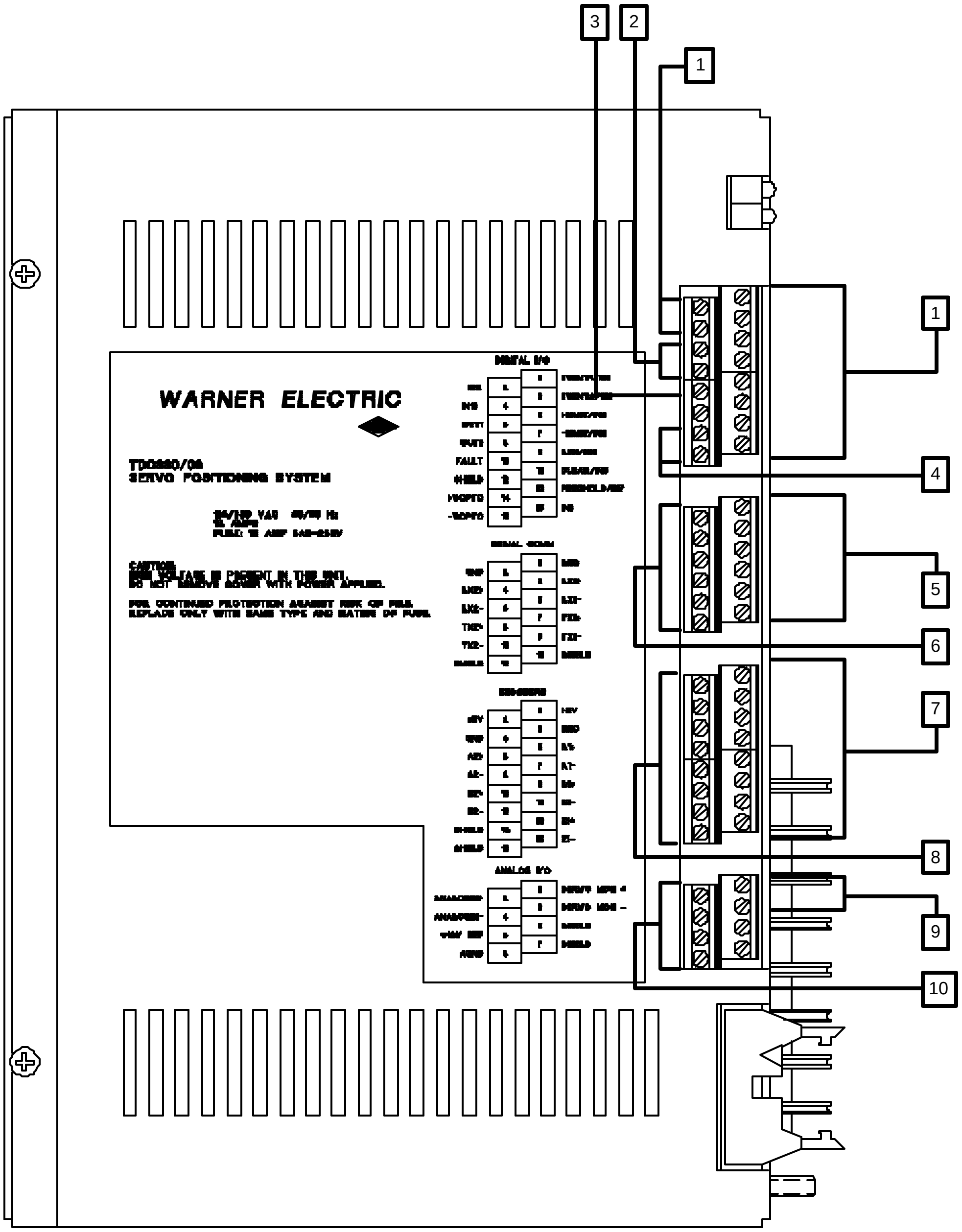


Figure 3.3
TDC Connections – Side View

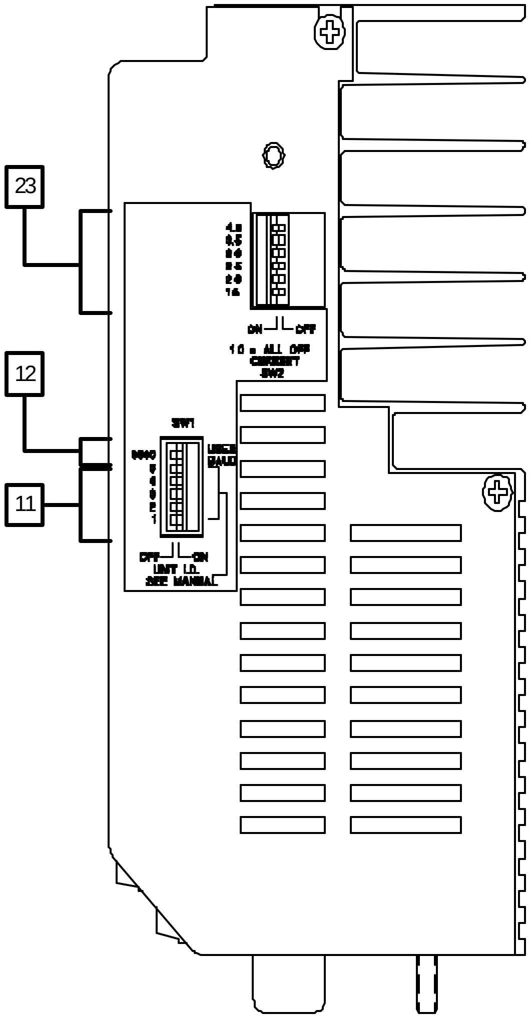


Figure 3.4
TDC Switches

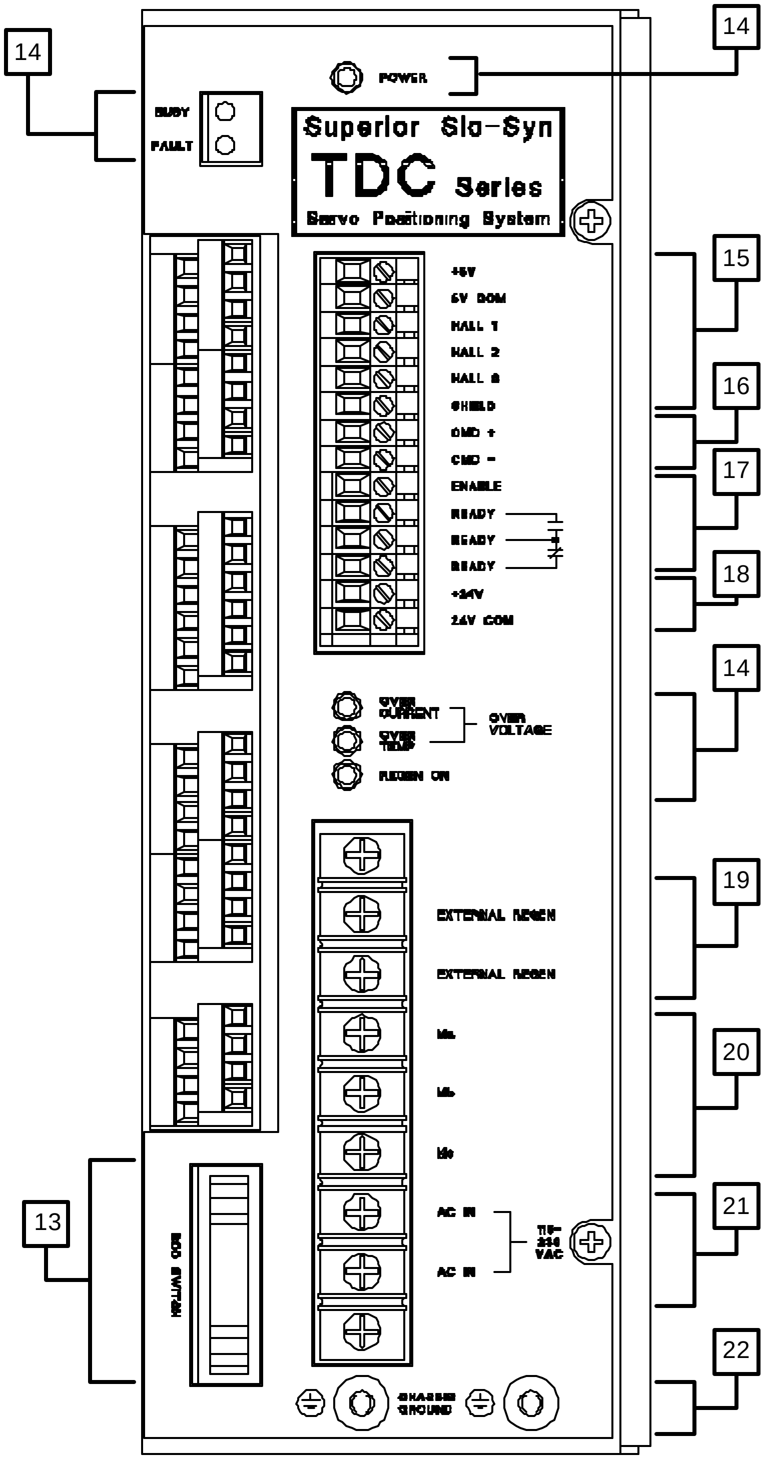


Figure 3.5
TDC Connections Front View

1 Inputs 1-10

EVENT 1/ IN1; EVENT 2 / IN 2

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

+LIMIT / IN3; -LIMIT / IN4

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

RUN / IN5

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

CLEAR / IN6

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

FEEDHOLD / IN7

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

IN 8, IN9, IN10

These inputs can be used as programmable inputs.

2 Outputs 1 & 2

OUT 1, OUT 2

These outputs can be used as programmable outputs.

3 Fault Output

FAULT

FAULT is an output that is active low when a fault is indicated. The condition of the fault can be queried through the software **ERR** command.

4 OPTO

+VOPTO; -VOPTO

A power supply for the optical isolators is **REQUIRED** *for proper I/O operation. This supply must be connected to the +VOPTO and -VOPTO pins.* The +24VDC and +24V COM power supply is available from the drive connector on the TDC unit, and should be connected to +Vopto and -Vopto unless the user is to supply power for the I/O from a different source.

5 Serial Port 1

Host Serial Communications: Port 1

GND:	Ground for Serial Port 1
RX1+:	Receive+ for Serial Port 1 (RS485)
RX1-:	Receive- for Serial Port 1 (RS485/ RS232)
TX1+:	Transmit+ for Serial Port 1 (RS485/ RS232)
TX1-:	Transmit for Serial Port 1 (RS485)
Shield:	Connection for Shield

6 Serial Port 2

Auxiliary Serial Communications: Port 2

GND:	Ground for Serial Port 2
RX2+:	Receive+ for Serial Port 2 (RS485)
RX2-:	Receive- for Serial Port 2 (RS485)
TX2+:	Transmit+ for Serial Port 2 (RS485)
TX2-:	Transmit- for Serial Port 2 (RS485)
Shield:	Connection for Shield

7 Encoder 1

ENCODER 1

Encoder inputs for the servo motor can be single-ended or differential phase quadrature.

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

8 Encoder 2

+5V: +5V supply for encoder.
GND: Ground for encoder.
A2+: Encoder Channel A+ input.
A2-: Encoder Channel A- input.
B2+: Encoder Channel B+ input.
B2-: Encoder Channel B- input.
Shield: Connection for Shield
Shield: Connection for Shield

9 Servo Monitor Outputs

SERVO MON OUTPUTS
The servo monitor output is an analog signal proportional to the command to the drive or current amplifier. The monitor signal is -10 VDC to +10 VDC. **These signals are for connection to monitoring and measurement equipment ONLY. DO NOT connect them as input signals to any other type of equipment.**

SERVO MON +: Analog monitoring signal representing the current command from the controller.

SERVO MON -: Ground reference

10 Analog Input

ANALOG Input Connections
The analog input connections allow a voltage from -10 VDC to +10VDC to be read into the unit. Resolution of the input is 19.53 millivolts.

ANALOG IN+: Non-inverting analog input.
ANALOG IN-: Inverting analog input.
+10V REF: +10 VDC supply for precise referencing of analog signals.
AGND: Ground for analog inputs.

11 Device ID Number Switch

The DIP switches will allow up to 32 devices to be daisy chained together.

12 Baud Rate Switch

This switch is read only at power-up or after a reset command. In the **off** position the baud rate is forced to 9600. In the **on** position the baud rate for the loaded project is used. The User Baud is selected in the project *Configuration and Setup*. If no user program is loaded the default 9600 baud rate is used.

13 BCD Port

BCD Port / I/O
This port can be used as either a BCD port, consisting of 7 numbers and a sign **(Warner Electric Part # 221157-002)**, or used for additional outputs and inputs*.

BCD0 / IN11: BCD switch data 0 or program input 11.

BCD1 / IN12: BCD switch data 1 or program input 12.

BCD2 / IN13: BCD switch data 2 or program input 13.

BCD3 / IN14: BCD switch data 3 or program input 14.

BCD4 / IN15: BCD switch data 4 or program input 15.

BCD5 / IN16: BCD switch data 5 or program input 16.

BCD6 / IN17: BCD switch data 6 or program input 17.

BCD7 / IN18: BCD switch data 7 or program input 18.

BCD_STR0 / OUT3: BCD switch Strobe 0 or output 3

BCD_STR1 / OUT4: BCD switch Strobe 1 or output 4

BCD_STR2 / OUT5: BCD switch Strobe 2 or output 5

BCD_STR3 / OUT6: BCD switch Strobe 3 or output 6

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

14 LED's

These LED=s show conditions that may be occurring in the amplifier

BUSY: Signifies that motion is occurring on the motor.

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

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

OVER CURRENT: Shows that current above the peak current is being drawn. Could be a phase to phase or phase to ground short. May also be caused by a non-compatible motor being used.

OVER TEMP: This LED shows that the temperature of the heat sink is in excess of 70 EC or the ambient temperature is greater than 50 EC.

OVER CURRENT & If the over current **and** the **OVER TEMP**: over temperature LED=s are active this indicates that an over voltage condition has occurred, bus voltage > 425VDC.

REGEN ON: Shows that the regenerative energy circuitry is activated due to energy.

15 Hall Effect Sensors

These connections are used to power and connect the Hall effect sensors to the amplifier, and are optically isolated inputs for the commutation devices. The inputs are pulled up to a +5 V supply through 1000 ohm resistor. Low level current is < 4 milliamperes.

+5V: +5VDC to power the Hall effect sensors.
5V COM: 5VDC common.
HALL1: Hall Effect sensor input for phase one(1).
HALL2: Hall Effect sensor input for phase two(2).
HALL3: Hall Effect sensor input for phase three(3).
SHIELD: Shield connection.

16 CMD+,CMD-

These signals should not be connected on TDC units.

17 Enable and Ready signals

ENABLE: Sinking this input will enable the amplifier. The amplifier will only enable if the controller allows it and the user activates the input by sinking current to **5VCOM** with a switch or open collector active device.

READY: The READY output is active when the amplifier is ready to accept a servo command from the controller. Both normally open and normally closed relay contacts are available on the drive connector.

18 24 VDC Power Supply

24 VDC to power other devices. It is recommended that this power be connected to the +Vopto and -Vopto in puts on the controller as the discrete I/O supply.

+24V: +24VDC output at 0.5 Amps.
24V COM: Common for the 24VDC output.

19 External Regen Resistor

Allows for the connection of additional regeneration resistors. External Regeneration resistor(s) may be required if the system inertia and deceleration requirements exceed specified limits. Example energy and power calculations to aid in sizing external regeneration resistors are given in Section 3.6.

20 Motor Wiring

These connections are for connecting the three phases of the motor.

Ma: Motor phase a.
Mb: Motor phase b.
Mc: Motor phase c.

21 AC Power

AC CONNECTIONS
These inputs are for connection of single phase AC power. The input power range is from 94VAC to 264VAC

22

Chassis Ground

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

Current Settings

The current settings of the drive reflect the continuous current capability of the motor. Peak current is twice the continuous current setting. The TDC-04 has a maximum continuous current of 4 amps and a peak current of 8 amps. The TDC-08 has a maximum continuous current of 8 amps with 16 amps peak current. The highest DIP switch setting in the on position will be the max. **CONTINUOUS** current supplied by the drive.

3.5 Wiring Diagrams

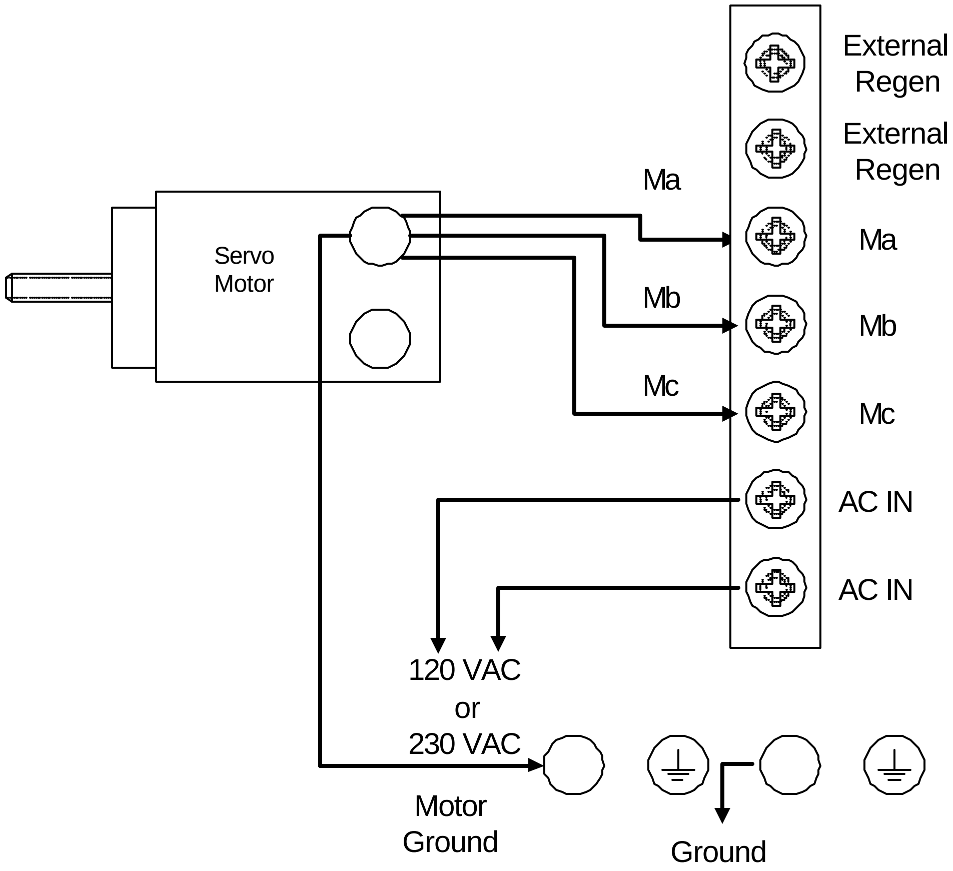
This section provides diagrams for wiring each connection discussed above. Remember to follow the General Wiring Guidelines outlined previously.

Caution Warning

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

3.5.1 Motor Connections to the amplifier.

A. Connect **phase A** of the motor to the terminal marked **Ma**.
B. Connect **phase B** of the motor to the terminal marked **Mb**.
C. Connect the motor **phase C** to the terminal marked **Mc**.
D. Connect the **motor ground** **to the ground stud below the terminal strip**.



Motor & Hall Sensor Commutation Table

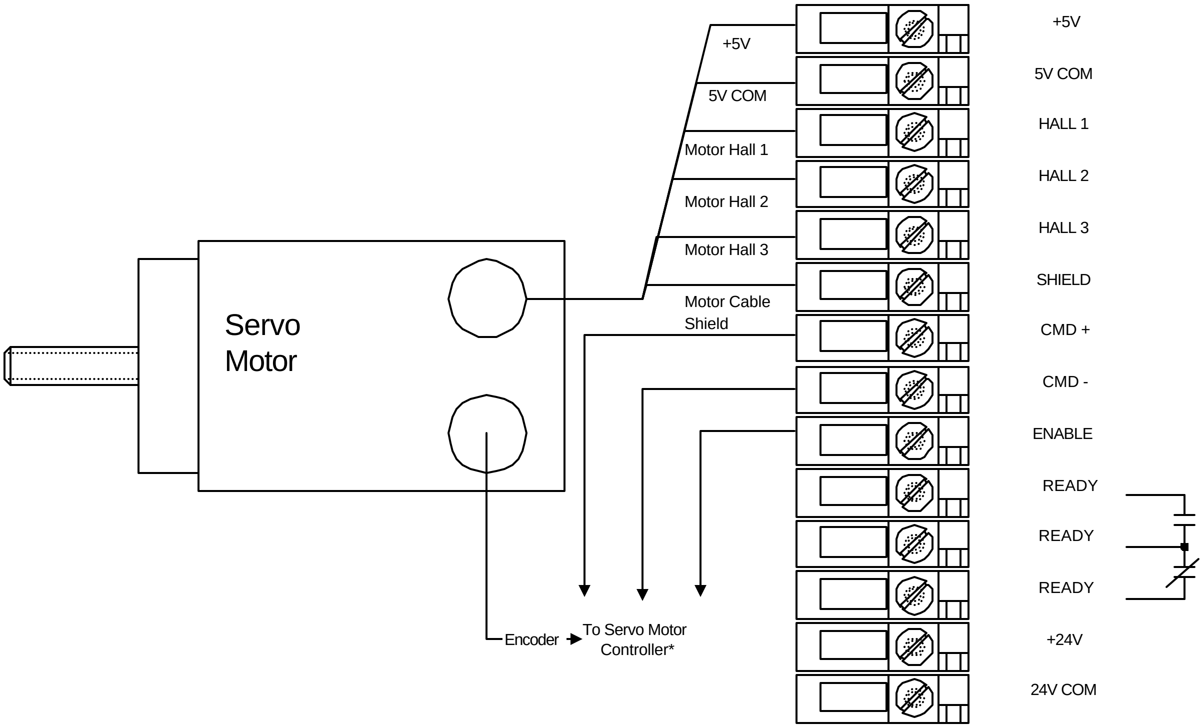
HALL 1	HALL 2	HALL 3	MOTOR VOLTAGE
+5 VDC	0V	0V	Ma to Mb
0V	+5VDC	0V	Mb to Mc
0V	0V	+5VDC	Mc to Ma

The above table must be used to properly Aphase@ your motor=s windings to the three hall effect sensors or encoder commutation tracks.

Note: A pullup resistor of 1Kohm to +5VDC is provided on the three HALL inputs internal to the drive unit. If an open-collector hall sensor is used, the device must be off to yield a +5VDC signal at the drive HALL terminal.

3.5.2 Hall Sensors / Commutator connections

A. Connect the **5V** and the **5V COM** to the amplifier.
B. Next, connect the **phase A hall sensor or commutator to Hall 1** on the amplifier.
C. Connect the **phase B hall sensor or commutator to Hall 2** on the amplifier.
D. Connect the **phase C hall sensor or commutator to Hall 3** on the amplifier.
E. Connect the **shield of the cable to the Shield input** on the amplifier.



*Connection of the Servo CMD+ or CMD- depend on servo contoller used. If used with the "TDC", CMD+, CMD- and ENABLE are monitoring points only.

The CMD+, CMD- inputs should NOT be connected on TDC units. These are connected internally. Connection to these terminals will cause improper operation of the unit.

3.5.3 Servo Motor Encoder connections

3.5.3.1 Encoder 1

Connections for Encoder 1 should be made according to figure 3. The Z-channel may also be known as the Index channel or I-channel. Single ended TTL encoder channels should be connected to the “+” inputs.

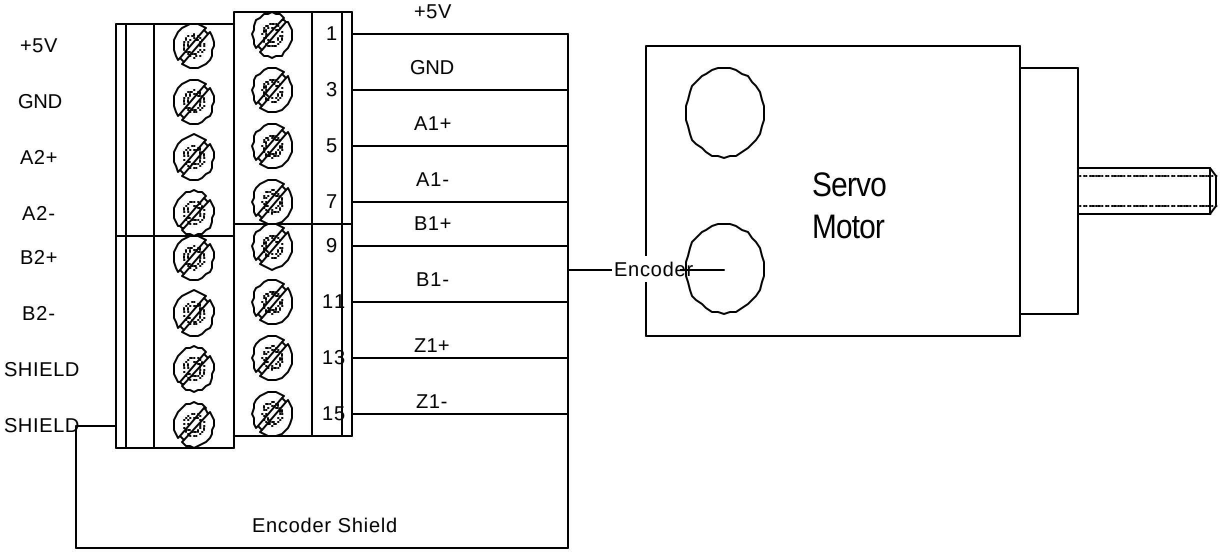


Figure 3.8a, Encoder 1 Connections

3.5.3.2 Encoder 2

The encoder 2 connector is used to connect an external quadrature encoder source to the controller. **Encoder 2 line count** in the configuration **Encoder folder** represents the number of lines in one revolution of encoder 2. Position and velocity of encoder 2 may be read using the **ENCPOS2** and **ENCSPD2** commands. In electronic gearing mode encoder 2 may be selected as the master source by using the **GEAREXT and GEARON** commands**.** The motor (slave) to master ratio is set using **GEARRATIO** .The external quadrature encoder connections are depicted in the Figure 3.8b. Although differential encoder outputs are recommended, single ended TTL encoder channels can be connected to the A+, and B+ terminals.

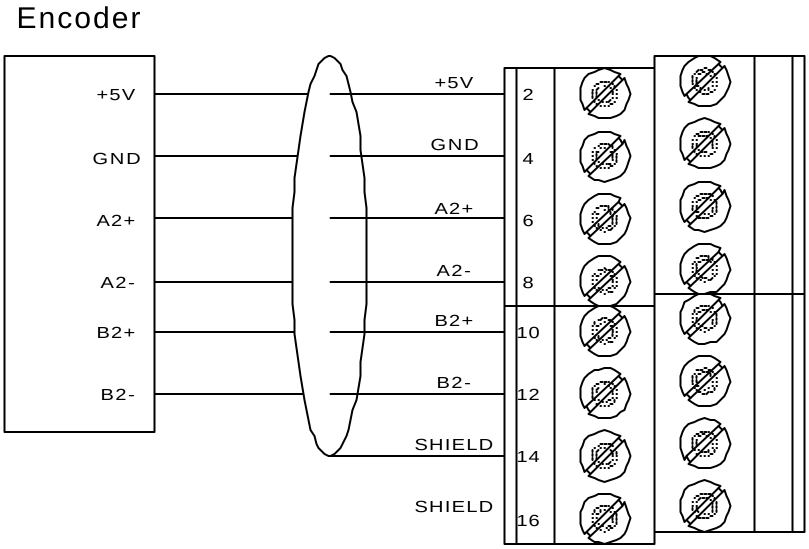


Figure 3.8b, Encoder 2 Connections

3.5.4 Input Connections

Inputs can be sink or source using active devices or a switch closure. Connections to the inputs can be seen in figure 4. Connections to other inputs are similar.

The factory setting for inputs 1-10 is current sinking. Changing the setting to **sourcing** requires the removal of the steel enclosure to expose the circuit card. Refer to section 3.1.1 of this manual.

Note: **The BCD port can be configured as TTL sink only I/O. See figure 6 and the Electrical Specifications section for more detailed information.**

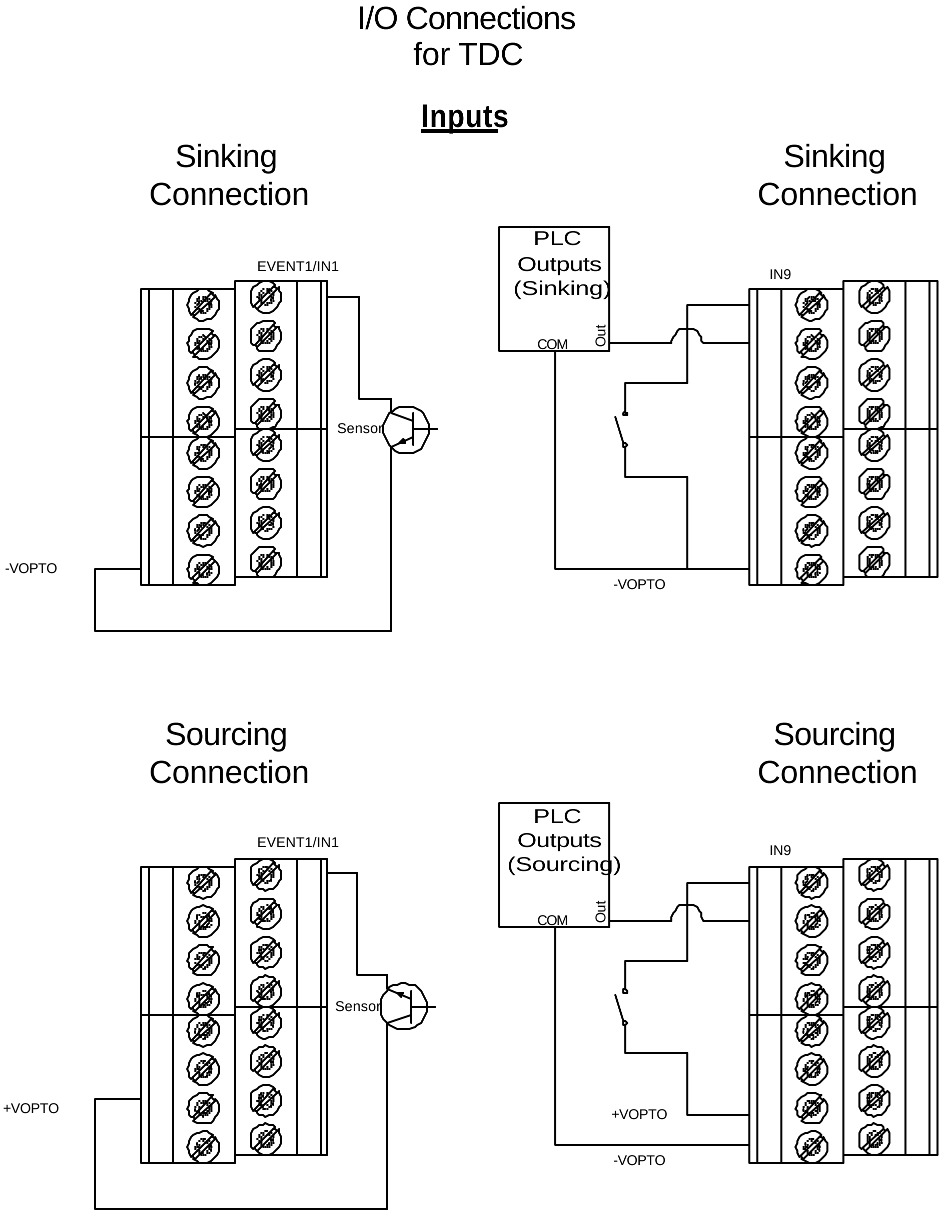


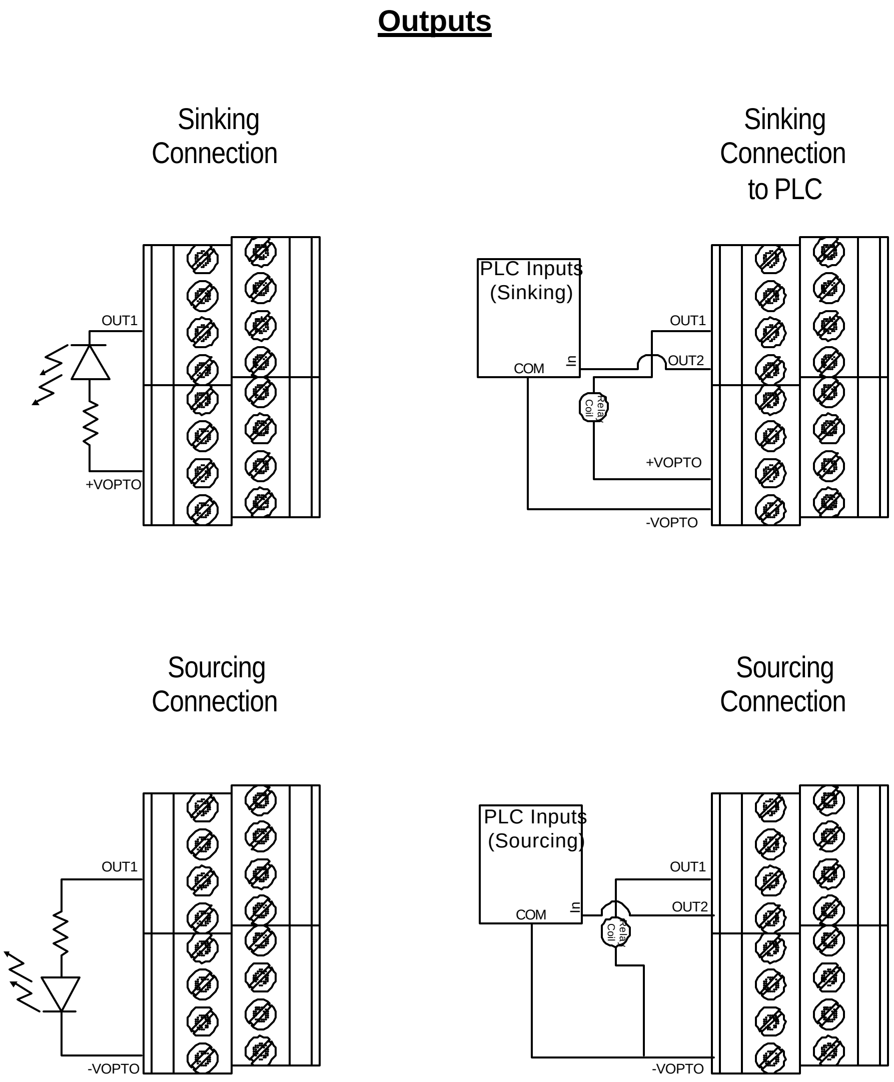
Figure 3.9
Digital Input
Connections

3.5.5 Output Connections

Outputs can be sink or source. Connections to the outputs is shown in figure 5. Connections to other outputs is similar.

The factory setting for the outputs is current sinking. Changing the setting to **sourcing** requires the removal of the steel enclosure to expose the circuit card. Refer to Section 3.1.1 of this manual.

Note: The BCD port can be configured as TTL sink only I/O. See figure 6 and the Electrical Specifications section for more detailed information.



Outputs

Sinking Connection

Sinking Connection to PLC

Sourcing Connection

Sourcing Connection

Figure 3.10
Digital Output Connections

BCD / TTL
I/O

BCD Strobe 2 TTL Out 5	1	2	BCD Stobe1 TTL Out 4
BCD Strobe 3 TTL Out 6	3	4	BCD Strobe 0 TTL Out 3
BCD Data 6 TTL IN 17	5	6	BCD Data 7 TTL IN 18
BCD Data 4 TTL IN 15	7	8	BCD Data 5 TTL IN 16
N.C.	9	10	N.C.
BCD Data 3 TTL IN 14	11	12	BCD Data 0 TTL IN 11
BCD Data 2 TTL IN 13	13	14	BCD Data 1 TTL IN 12

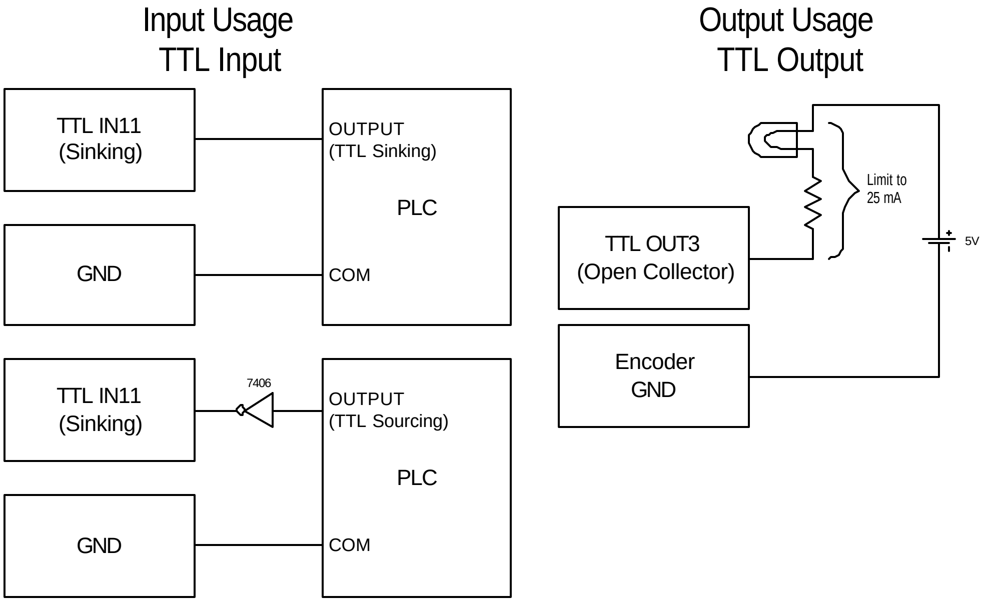


Figure 3.11
BCD/TTL Digital
I/O Connections

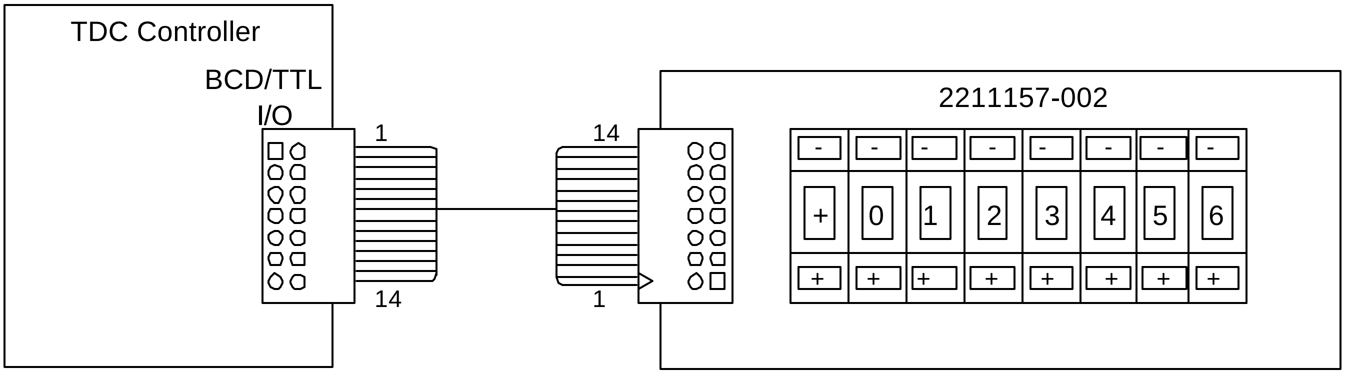


Figure 3.12a
BCD Switch Connections

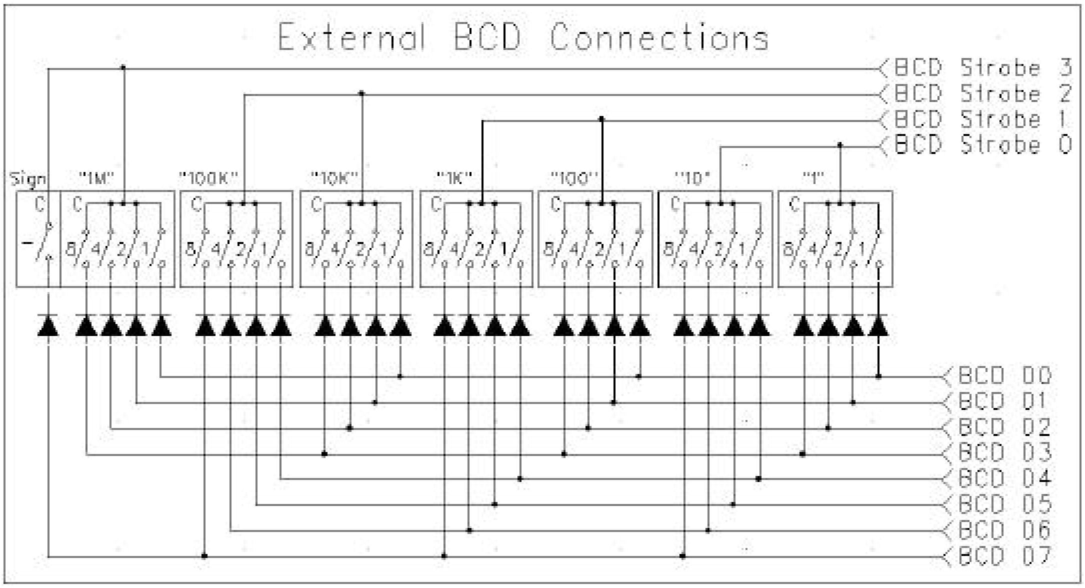
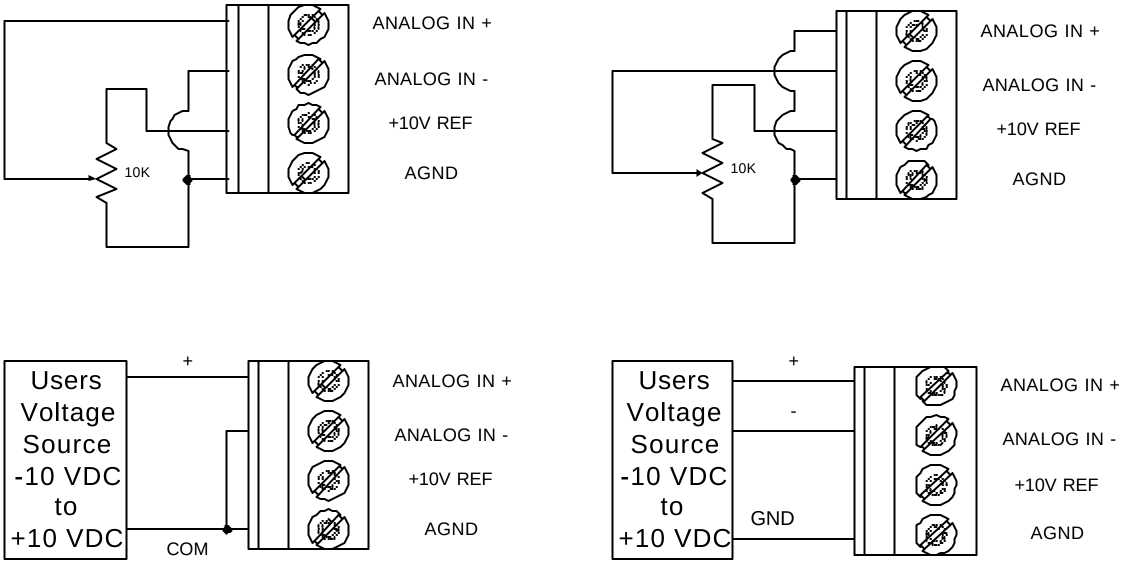


Figure 3.12b
Typical External BCD Connections

3.5.6 Analog Input Connections

The analog input can accept a -10 VDC to a + 10 VDC signal. This input can be single ended or differential. Examples of connection methods are shown in figure 8. A +10V REFerence supply capable of sourcing 10mA of current is also provided.

The user's ground must be referenced to the **GND** or **AGND** on the controller.



Single Ended Analog Signal

Differential Analog Signal

Figure 3.13
Examples of
Analog Input
Connections

3.5.7 RS232 / RS485 Host Serial Communication Connections

This serial port is for communication and programming of the controller from a personal computer (PC). The port can be configured for RS232 or RS485 operation. The factory setting is for RS-232. The port may be configured by removing the unit cover as described in section 3.1.1. The wiring connection diagram is shown below. Note that when wired for RS-485 operation, a cable with individual twisted pair wires must be used. The termination resistors indicated with the * must have a value of 120 ohms. The resistor across the RX1+ and RX1- lines must be added external to the TDC unit. The termination resistor across the TX1+ and TX1- signals at the **opposite** end from the TDC may also be required if the terminal device to which the TDC is communicating does not provide the termination resistor internally. If the terminal device does provide the resistor internally, the resistor across TX1+ and TX1- is not required. Check with the manufacturer of the terminal device and review its specifications to insure proper serial communications operation.

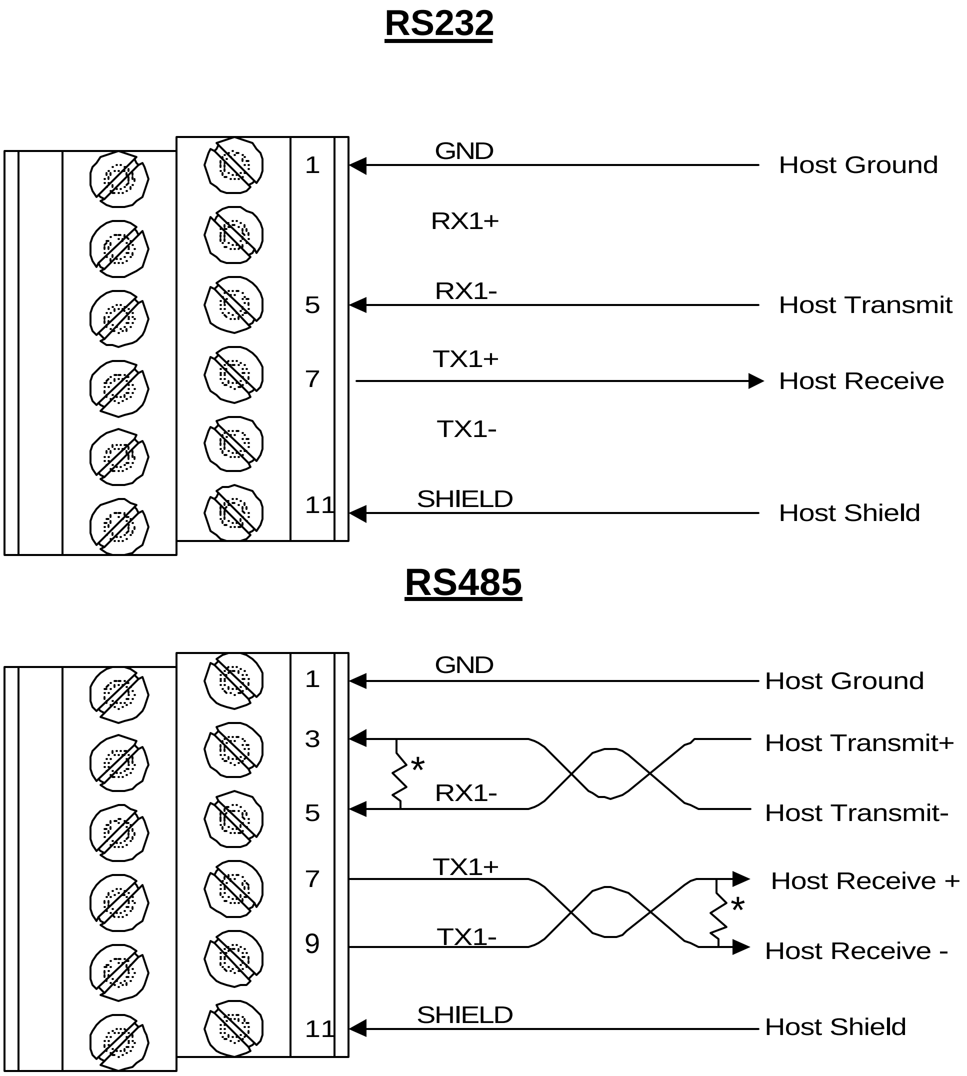


Figure 3.14
Host Serial Connections

3.5.7.1 - RS485 Host Daisy Chaining Connections

Connection in a daisy chain configuration requires that the Host port of all units be wired as RS-485. Each unit must also be set to RS485 Host communications mode. To change the Host port communications mode remove the 8 screws holding the cover on and place the jumpers on JP4 & JP7 to the RS485 position, see Figure 3.1. Put the cover back on and secure the 8 screws holding the cover in place. Be sure that the unit is off when changing the switch position.

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

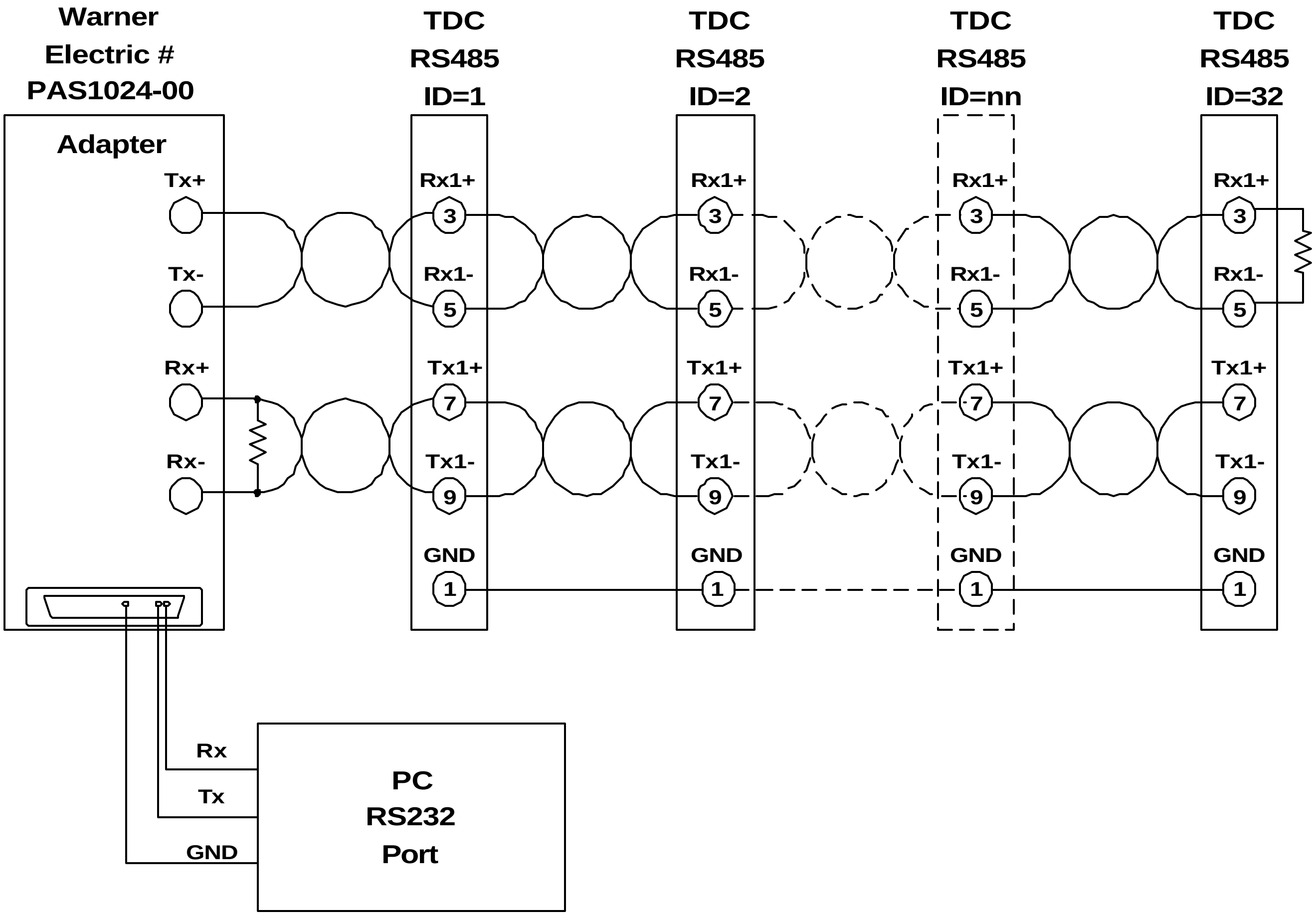


Figure 3.15
Daisy Chaining Wiring Diagram

3.5.8 RS485 Auxiliary Communication Connections

The auxiliary serial port is used for serial communication to and from other devices, such as PLC=s or operator interface panels. This serial port is RS485 only. The wiring connection diagram is shown below. Note that a cable with individual twisted pair wires must be used. The termination resistors indicated with the * must have a value of 120 ohms. The resistor across the RX2+ and RX2- lines must be added external to the TDC unit.

The termination resistor across the TX2+ and TX2- signals at the **opposite** end from the TDC may also be required if the terminal device to which the TDC is communicating does not provide the termination resistor internally. If the terminal device does provide the resistor internally, the resistor across TX2+ and TX2- is not required. Check with the manufacturer of the terminal device and review its specifications to insure proper serial communications operation.

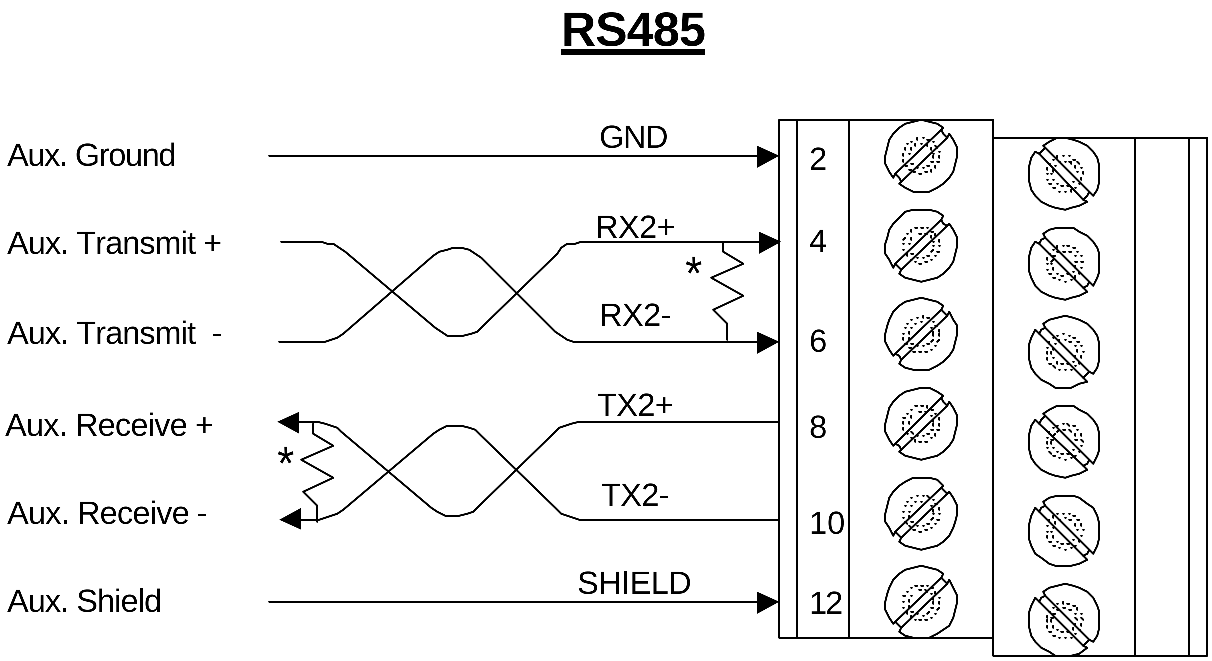


Figure 3.16
Auxiliary
Serial Connections

3.5.9 AC Power Connections to the Unit

Connect the two (2) AC IN terminals to the line voltage. The voltage input can be from 95 VAC to 264 VAC 50/60 hertz. The voltage in will determine the DC bus voltage. The DC bus voltage can be approximated by using the following equation:

$$VDC_{OUT} = VAC_{IN} \text{ x} \sqrt{2}$$

$$VDC_{OUT} = VAC_{IN} \text{ x } 1.4142$$

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

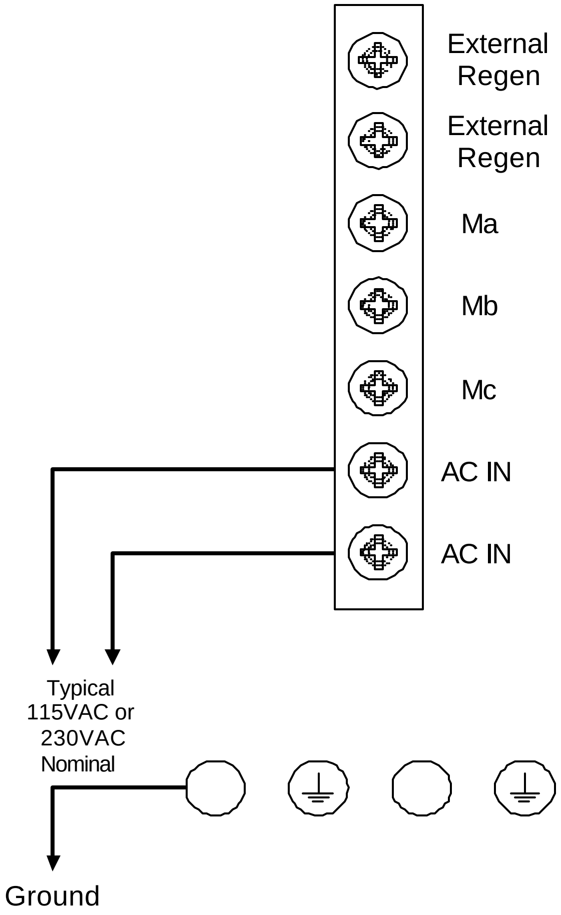


Figure 3.17
AC Power Connection

3.5.10 Regenerative Resistor Connections

An external regenerative resistor can be connected to the 7 position terminal strip connection points marked external regen . Refer to Figure 3.5 item 18 and Figure 3.17. See the following example to determine if an external regenerative resistor is required.

For determining if the internal regenerative resistor is sufficient use the following formula:

J_{rotor} = Inertia of the rotor in lb-in-sec^2
J_{load} = Inertia of the load in lb-in-sec^2
P = Power of regen resistor in watts
N = Speed of the motor in RPM
T_{frict} = Frictional torque in lb-in
t_{decel} = Deceleration time
t_{cyc} = Cycle time
E = energy in motor in joules

$$E=\frac{1}{1616} * (J_{rotor}+J_{load}) * N^2 - \frac{1}{85} * T_{frict} * N * t_{decel}$$

$$t_{cyc} = \frac{E}{P}$$

where: **P** = Power of the internal resistor = 50 W

The internal resistor is sufficient if the calculated energy is # 200 Joules **and** the cycle time is greater than the calculated cycle time.

Example:
J_{rotor} =.0026 lb in sec^2
J_{load} = 0.130 lb in sec^2
P = 50 W
N = 4000
T_{frict} = 5 lb. in.
t_{decel} = .25 sec

$$E= \frac{1}{1616} * (.0026 \ +.0130) * (4000)^{2} - \frac{1}{85} * 5 \ * 4000 * .25 = 95.63$$

$$t_{cyc} \geq \frac{95.63}{50} \geq 1.9$$

Since the energy is less than 200 joules and if the cycle time is greater than 1.9 seconds no external regen is required. If the energy is greater than 200 joules or the cycle time is less than the calculated cycle time consult the factory for the proper external regen resistor.

Section 4

Hardware Specifications

4.1 Mechanical and Environmental Specifications

Size:	3.76W x 10.63H x 8.07D (See Figure 3.2)
Operating temperature:	+321 F to +1221 F (01 C to +501 C)
Storage temperature:	-401 F to +1671 F (-401 C to +751 C)
Humidity:	95% maximum, non-condensing
Altitude:	10,000 feet (3048 meters) maximum

4.2 Electrical Specifications

Input Voltage:	Single Phase 95 to 264 Vac, 50/60 Hz
Input Current:	TD330/04, 7 amps, internal fuse rating: **8 amp 3AB Fast Acting**
	TD330/08, 14 amps, internal fuse rating **15 amp 3AB Fast Acting**
Internal Bus Voltage:	130 to 375 Vdc depending on AC Input Voltage
Motor Current:	TD330/04, 4 amps continuous, 8 amps peak
	TD330/08, 8 amps continuous, 16 amps peak
	(maximum value of a 6-step waveform)
Regenerative Energy Circuit:	Internal: 50 ohm, 50 watt resistor
	External: Connections available (Must be externally fused)
	See example for size selection in Section 3.5 or consult factory
Regenerative Energy Circuit Fusing:	Internal fuse, **1 amp 3AG Slo-Blo**

4.2.1 Isolated Digital I/O

Inputs (**IN1 - IN10**): Sink or Source mode selectable via jumper JP3.

Sink mode:

On state voltage range (V_IL) with -Vopto = 0V:	**0V to (+Vopto - 4.5V)**
Input Current; (V_IL = 0V), +Vopto=24V, -Vopto=0V:	**11.5mA**
Input Current; (V_IL = 19.5V), +Vopto=24V, -Vopto=0V:	**1.7mA**
Maximum Input differential voltage, (+Vopto - V_IL):	**26V**

Source mode:

On state voltage range (V_IH) with -Vopto = 0V:	**4.5V to 26V**
Input Current; (V_IH = 4.5V) with -Vopto=0V :	**1.7mA**
Input Current; (V_IH = 26V) with -Vopto=0V :	**11.5mA**
Maximum Input differential voltage, V_IH-(-Vopto):	**26V**

Response time (sink or source):

Opto turn on delay:	**10uS typical**
Opto turn off delay:	**75uS typical**

Programmable Outputs (**OUT1,OUT2**): Sink or source selectable via jumpers JP1 and JP2.

Sink mode:

Current rating:	**50mA continuous.**
Maximum collector voltage with -Vopto = 0V:	**26V max**
On state voltage @ 50mA:	**2.0V max**

Source mode:

Current rating:	**50mA continuous.**
Maximum emitter voltage with -Vopto=0V:	**26V max**
On state voltage @ 50mA:	**+Vopto - 2.0V**

4.2.2 TTL I/O or BCD Interface, non-isolated*:*

IN 11- IN18**:**

Logic high input level: (Open circuit or sourcing voltage**)**	**5V > Vsource > 4.5V**
Logic low input level:	**1.5V max**

OUT 3 - OUT6:

These are open-collector, sink only TTL outputs which are NOT isolated from the unit:s +5 V logic supply. Proper care must be exercised to insure noise is not injected onto these signals.

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

FAULT Output***:***

The FAULT output is **SINK ONLY,** active low. The specifications are identical to OUT1 and OUT2 in sink mode as listed above.

4.2.3 Serial Communications:

Port 1:

Configurable for RS-232C or RS-485 specifications via jumpers JP4 and JP7. For RS-232 mode RX1- is used for receive data into the unit, and TX1+ is used as the transmit data out. Port 1 is designated as the HOST communications port.

Port 2:

Serial channel 2 is for RS-485 USER communications.

Serial Port Data Format:

Baud Rate:	**9600 – 38.4K**
Data Bits:	**8**
Stop Bits:	**1**
Parity:	**None**

4.2.4 Encoder 1 and 2 Connections:

Encoder channel 1 provides power and inputs for a digital encoder interface to provide servo motor position to the controller.

Encoder channel 2 provides power and input for an external encoder interface which can be used as an external master source for electronic gearing, or as a digital position input to a user program when gearing is not used.

Encoder +5Vdc power supply output:	**+5VDC (+/- 5%) @ 100mA current**.
Encoder signal inputs:	**TTL level single ended or differential channels A and B in phase quadrature. Single ended TTL signals, use A+, B+, and Z+ inputs. (Z+ applies to encoder 1 only)**
Input current A+,A-,B+,B-:	**+/- 5mA min**
Maximum Frequency:	**500 Khz per channel @ 2 ms sample time** **250 Khz per channel @ 1 ms sample time**

4.2.5 Analog Input*:*

Voltage Range:	**+10V(max) to -10V (min) referenced to AGND**
Resolution:	**10 bits or 19.5mV**
Absolute Accuracy:	**+/- .3V worst case**
Sample Rate:	**500 Hz min**
Bandwidth:	**100 Hz max**

4.2.6 Analog +10V Reference:

Voltage:	**+10.00 (+/- 20mV)**
Output current source rating:	**10mA max**

4.2.7 Servo Monitor Output:

Voltage Range:	**+10V to -10V referenced to AGND**
Output Impedance:	**200 ohm max**
Output Current:	**5mA max, recommended sourcing to impedance > 20Kohm**

4.2.8 Internal Jumper Settings:

Also see section 3.1.1.

Serial Port 1 Select (JP4 and JP7s):

Serial port 1 may be configured for RS-232 or RS-485 differential mode operation. Power must be disconnected and the unit cover removed to access the jumpers located on the controller circuit board. Jumpers JP4 and JP7 must be set to either the RS232 or RS485 positions as indicated on the circuit board. **BOTH jumpers MUST be installed in the same mode for proper operation of serial port 1.**

Inputs Sink/Source Select (JP3):

Jumper JP3 selects whether the discrete user inputs operate in either sink or source mode. In sink mode the opto couplers are internally tied to +Vopto, and the user must provide a sinking connection to -Vopto to activate the input (note reverse polarity of the CLEAR input). In source mode the opto couplers are internally tied to -Vopto, and the user must source voltage and current, preferably from +Vopto, to activate the input.

OUT1 Sink/Source Select (JP2):

Jumper JP2 selects whether user programmable output OUT1 is a sinking or sourcing output. With JP2 jumpered between 1-2 and 3-4, the output operates as a **sinking** NPN transistor with its emitter tied to -Vopto, and its collector available at the output pin. See the circuit diagram below. A shunt diode is provided from the collector to +Vopto to prevent overvoltage conditions from damaging the transistor due to switching of inductive loads.

With JP2 jumpered from 2-3 and 4-5, the output operates as a **sourcing** NPN transistor with its collector tied to +Vopto and its emitter available at the output pin. See the circuit diagram below. A shunt diode is provided from the emitter to -Vopto to prevent undervoltage conditions from damaging the transistor due to switching inductive loads.

OUT2 Sink/Source Select (JP1):

Jumper JP1 selects whether user programmable output OUT2 is a sinking or sourcing output. With JP1 jumpered between 1-2 and 3-4, the output operates as a sinking NPN transistor with its emitter tied to -Vopto, and its collector available at the output pin. See the circuit diagram below. A shunt diode is provided from the collector to +Vopto to prevent overvoltage conditions from occurring due to switching of inductive loads.

4.2.9 - Hall Effect Inputs

These are optically isolated inputs for the commutation devices. The inputs are pulled up to the +5 Vdc supply.

Impedance to +5Vdc supply:	**1K ohm pullup resistor**
Low level current:	**4 mA maximum**
Low level voltage:	**1V maximum**

4.2.10 - Enable Input

This input **Must** be connected (sunk) to 5Vcom **and** the controller **Must** set WNDGS = 1 to enable the drive.

Low level current:	**7 mA maximum**

4.2.11 - Ready Output

This is a relay output with NO and NC contacts which activates when the drive is ready to be enabled.

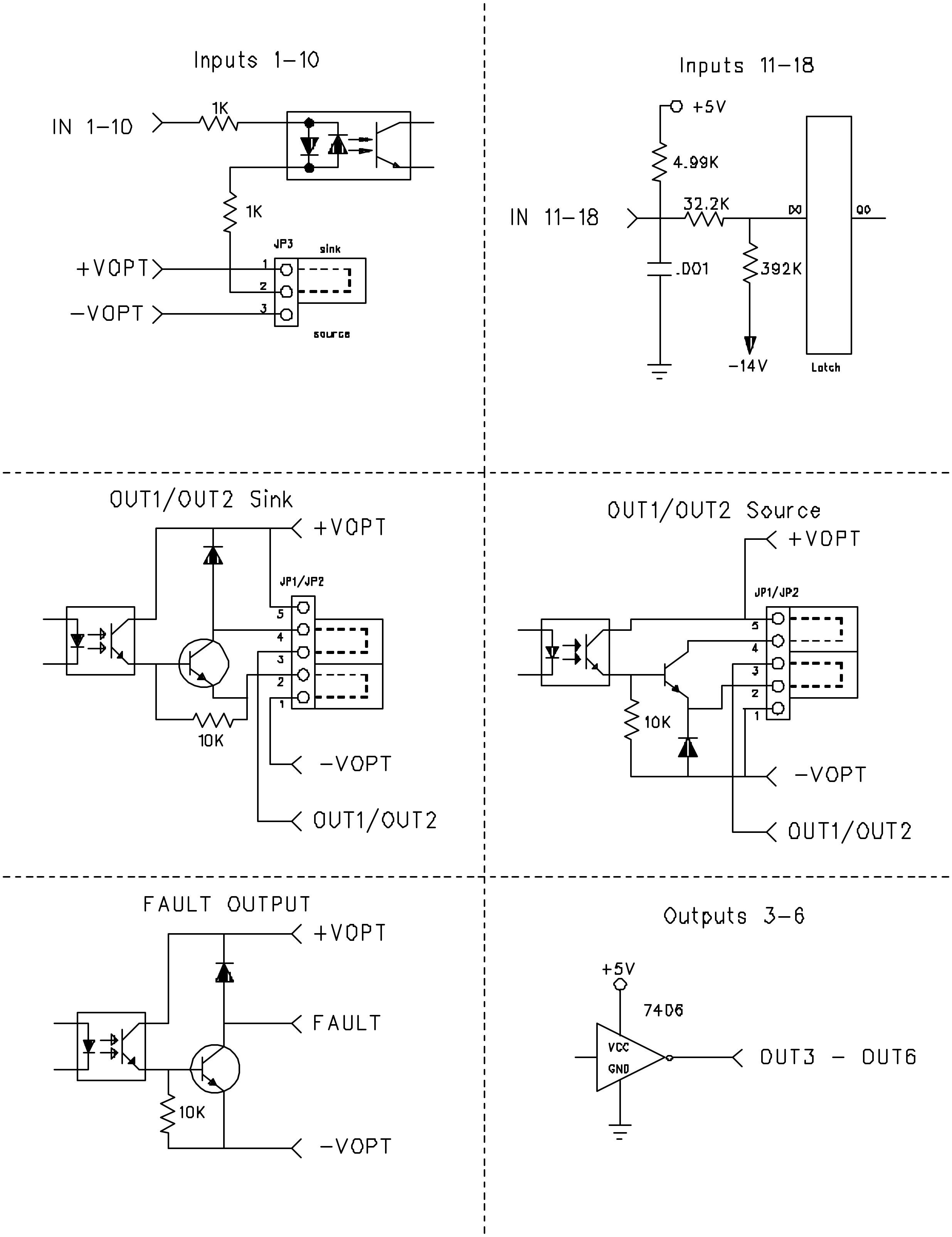
Contact rating:	**0.5A @ 24VDC resistive load**

4.2.12 – Motor Specifications

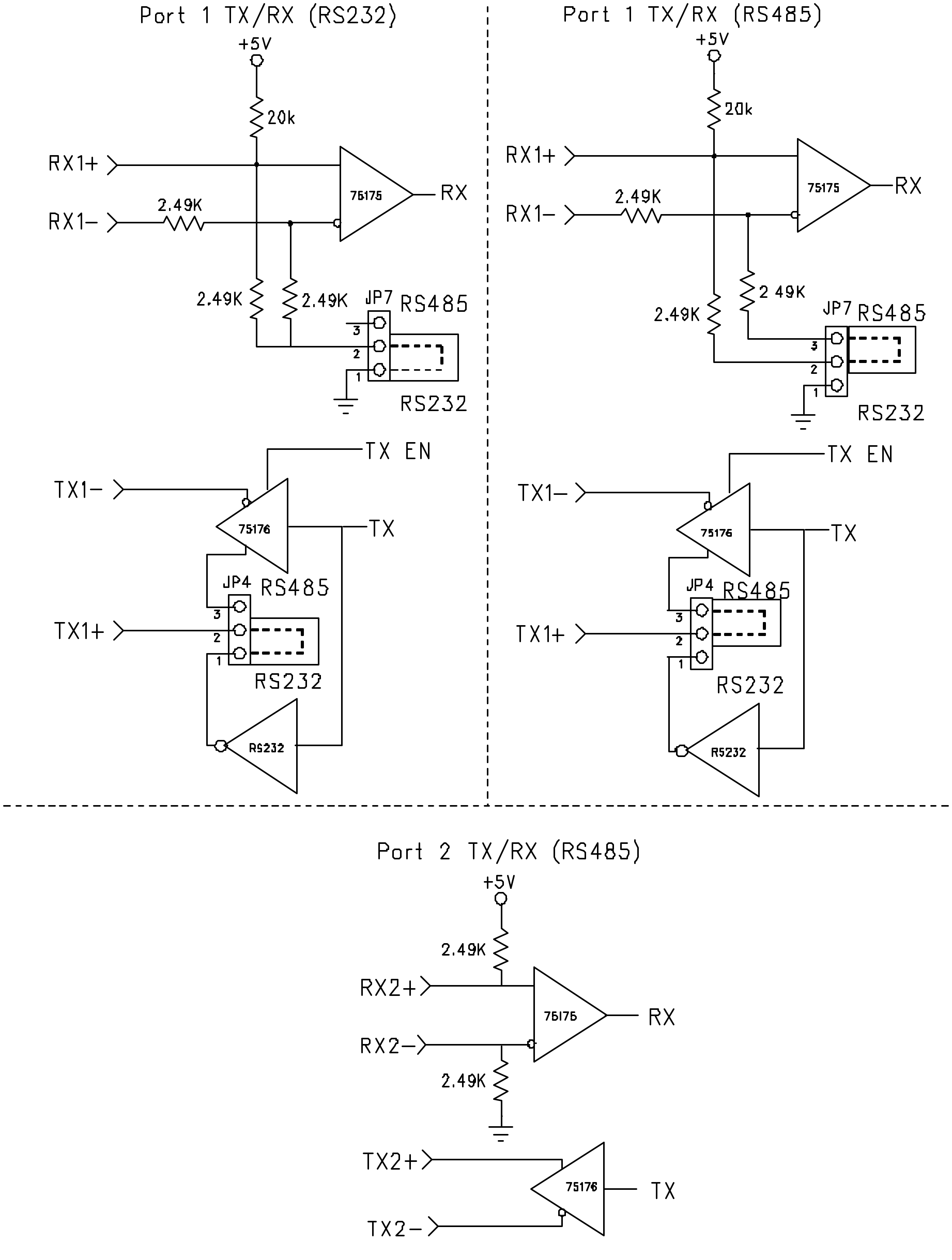
Type:		**3 Phase Brushless motor with Hall commutation or an encoder with commutation tracks. A 5V quadrature encoder is required for servo position feedback.**
Voltage:		**Cabable of withstanding bus voltages of 350 Volts.**
Current:	TDC330/04	**1 to 4 Amps continuous, 2 to 8 Amps peak**
	TDC330/08	**2 to 8 Amps continuous, 4 to 16 Amps peak**
Recommended Inductance:		**4 mH minimum**

4.3 - Hardware Equivalent Circuits

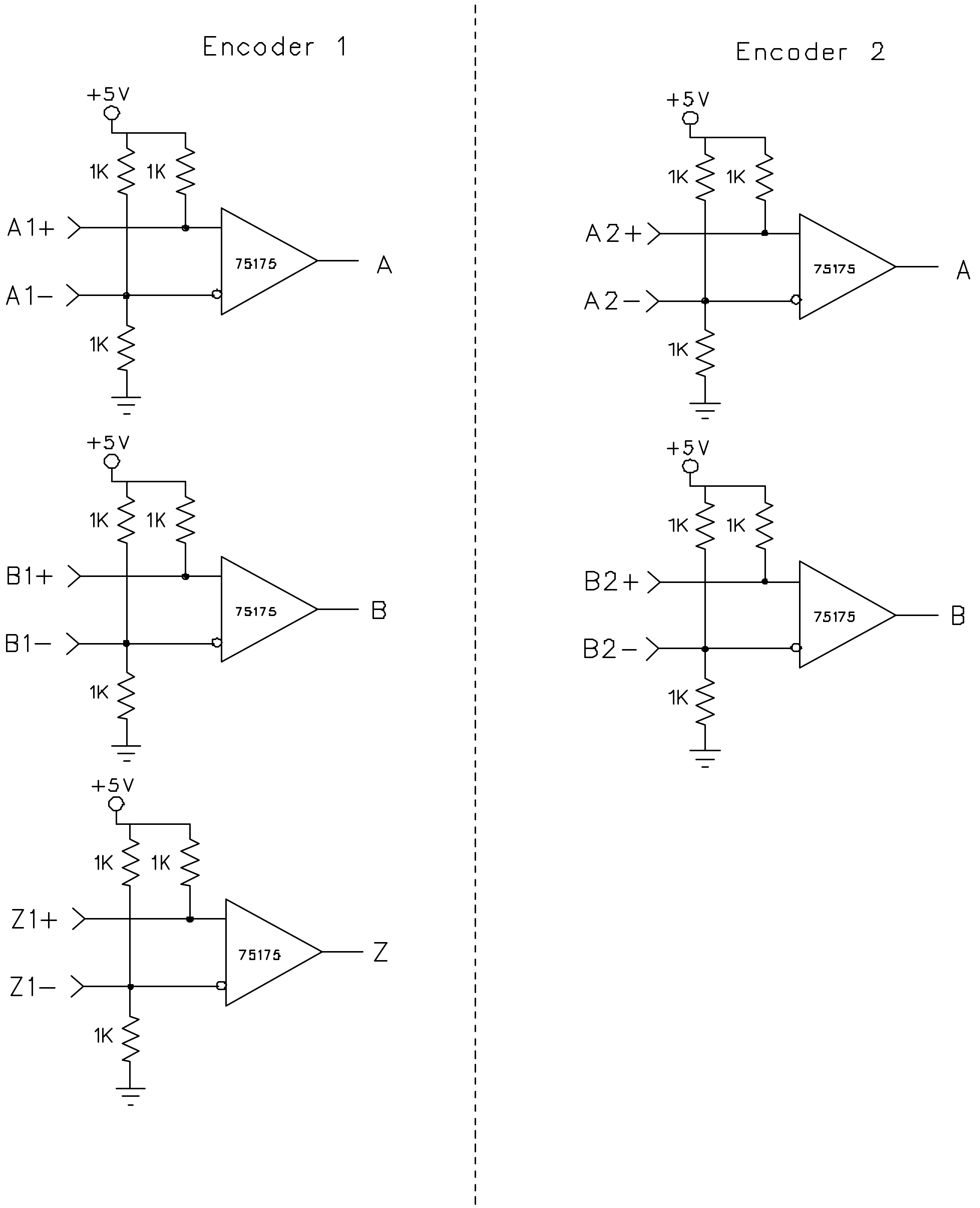
The following pages contain equivalent circuit diagrams which represent the physical interface hardware internal to the TDC unit. These circuits may be used as a reference if questions arise during detailed system design and integration. These diagrams are intended to represent the TDC interface circuitry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



I/O Equivalent Circuits



Communication Equivalent Circuits



Encoder Equivalent Circuits

Section 5

Programming Environment

5.1 - PROGRAMMING

5.1.1 - GENERAL DESCRIPTION OF PROGRAMMING

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

5.1.1.1 - What is Programming?

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

5.1.1.2 - What's in a Program?

A program consists of many individual lines organized in a prescribed sequence. The TDC system uses an English language, BASIC-type computer programming language (SEBASIC).

This makes it easy and intuitive to write and read machine control programs. SEBASIC supports many higher level language features, such as statement labels, subroutines, for-next and do-while loops for program flow control. This makes it easy to write concise, well organized, easily debugged programs. Also, there are built in mathematics, Boolean functions and two dimensional array capability. Finally, the motion, I/O, and timing commands are easy to understand, remember and apply.

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

5.1.1.3 - How is the Controller Programmed?

The programming environment called 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.

5.1.2 -What are Host Commands?

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

5.1.3 - Memory Types and Usage

A program is stored in memory. There are two kinds of memory. RAM (Random Access Memory) is called **Volatile Memory** because when power is removed from the controller, all of the information in that memory are lost. The Controller stores the program variables in RAM.

The second kind of memory is **Non-Volatile Memory**, such as FLASH memory, EEPROM or BBRAM memory. The information stored in this type of memory is not lost when power is removed. FLASH memory is used by the controller for storing the Operating System as well as the User Program. Battery-backed ram (BBRAM) is used for NVR storage.

Up to 400 REAL and/or INTEGER data locations are available for storage. These locations can be accessed from either host or basic programming commands. A checksum is developed and stored for each stored NVR location. Thus, data integrity is checked each time an NVR location is read. If the data checksum does not match during an NVR read an **NVR data corrupt error** (138) is generated.

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

5.1.4 - How to organize your Project

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

A good program will consist of an initialization, main program, Interrupt routines, Subroutines and an Error Handler sections. The Interrupt routines, Subroutines and Error Handler sections are optional. A Program Development Block Diagram is depicted on the following page.

5.1.4.1 - Initialization Section

The variable names and data types(Integer, Integer Array, Real and Real Array) are defined. Also the condition's which will trigger the individual Interrupts (INTR1-INTR4) may be defined. These commands apply to the TDC controller only.

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

Example Initialization Section:

```
 INTEGER a,b(100),c(10,3)
 REAL       d,e(50),f(5,4)
 STRING    x$, y$, z$

 ›          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
```

Examples of Integer values are:	1, 2, 5, 100
Examples of Real values are:	2.45, 3.1415, 10.735
Examples of String values are:	Any ASCII character

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

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

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

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

5.1.4.2 - Main Program Section

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

5.1.4.3 - Interrupt Routines

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

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

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

5.1.4.4 - Subroutines

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

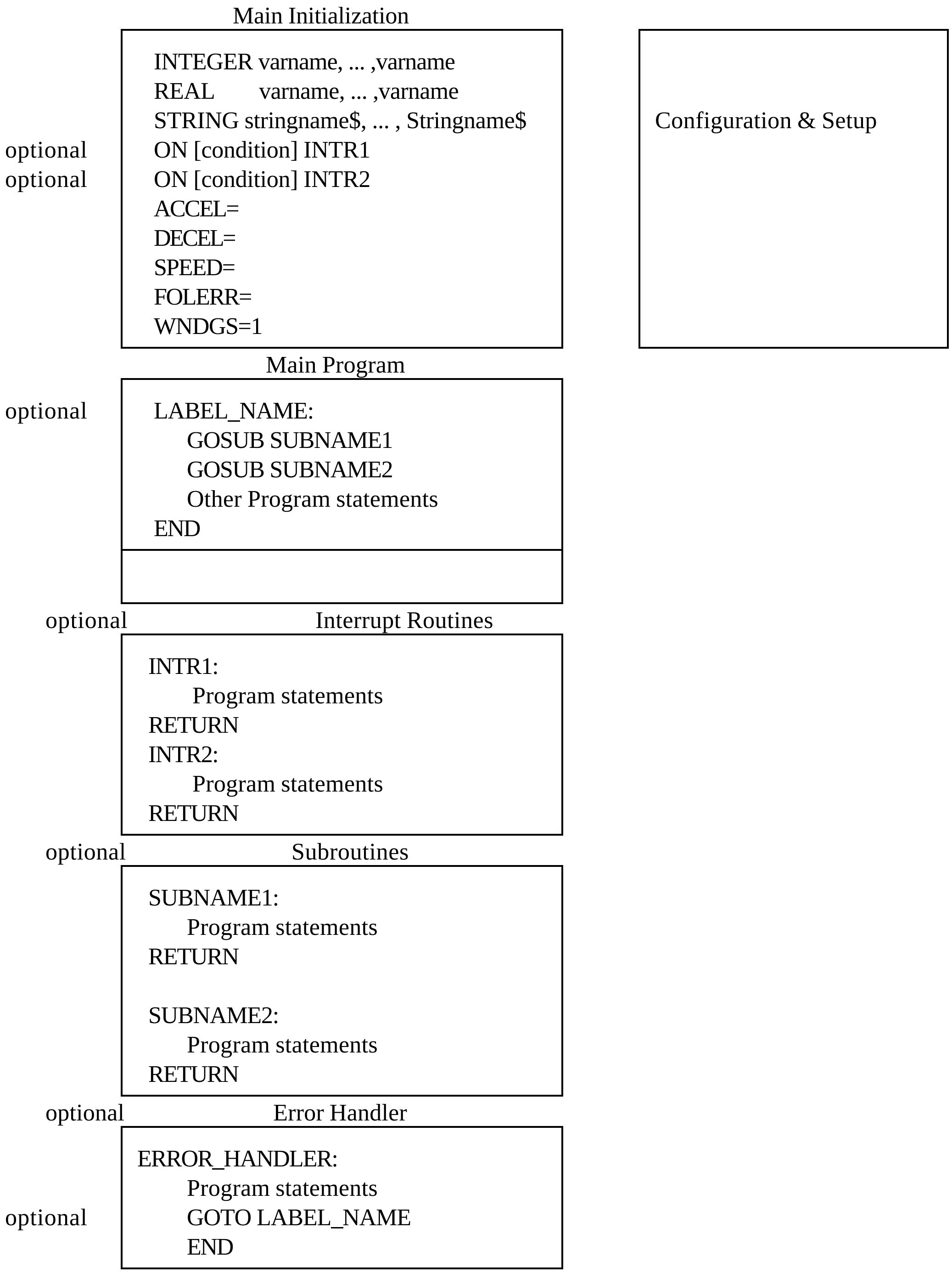
5.1.4.5 Error Handler

The Controller will handle error conditions during program execution in one of two ways:

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

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

Program Development Block Diagram



Main Initialization

```
          INTEGER varname, ... ,varname
          REAL       varname, ... ,varname
          STRING stringname$, ... , Stringname$
optional  ON [condition] INTR1
optional  ON [condition] INTR2
          ACCEL=
          DECEL=
          SPEED=
          FOLERR=
          WNDGS=1
```

Configuration & Setup

Main Program

```
optional  LABEL_NAME:
              GOSUB SUBNAME1
              GOSUB SUBNAME2
              Other Program statements
          END
```

optional Interrupt Routines

```
INTR1:
      Program statements
RETURN
INTR2:
      Program statements
RETURN
```

optional Subroutines

```
SUBNAME1:
     Program statements
RETURN

SUBNAME2:
     Program statements
RETURN
```

optional Error Handler

```
          ERROR_HANDLER:
              Program statements
optional      GOTO LABEL_NAME
              END
```

5.2 - MCPI PROGRAMMING ENVIRONMENT

5.2.1 - Software Installation

The MCPI programming environment 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 program. In addition, a **Terminal Mode** is provided for operating the controller from your computer.

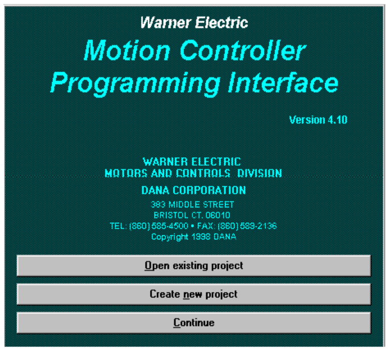
Installation Instructions:

1) If Windows 3.1(or 3.11) is not already running type **WIN** at the Dos prompt, and press **ENTER.**
2) Insert the TDC 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.

5.2.2 - Starting the programming environment

1) If the Windows 3.1 (or later) is not already running, type **WIN** at the DOS prompt, and press **ENTER**.
2) Double click on the MCPI Icon.
3) The opening screen will appear.

5.2.2.1 - The MCPI program opening screen



Warner Electric

Motion Controller Programming Interface

Version 4.10

WARNER ELECTRIC
MOTORS AND CONTROLS DIVISION
DANA CORPORATION
383 MIDDLE STREET
BRISTOL CT. 06010
TEL: (860) 585-4500 • FAX: (860) 589-2136
Copyright 1998 DANA

Open existing project opens up an existing project.

Create new project creates a new project.

Continue enters the MCPI Environment with no selection.

5.2.3 - Setting communication parameters

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

Computer	TDC
TX	RX-
RX	TX+
GND	GND

Notes: Consult your computer manual for the correct pin out of it's serial port. The factory default baud rate is 9600 baud. The TDC controller only supports the 9600, 19200 and 38400 baud rate.

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

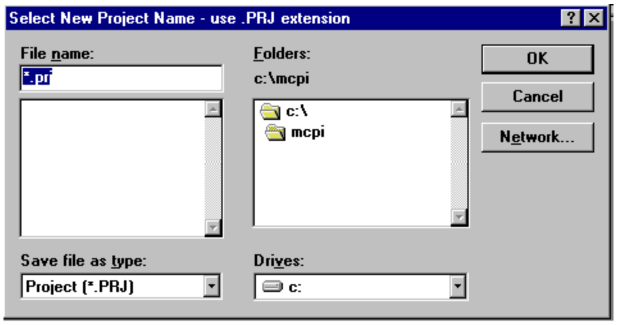
1) Set the 9600/User Baud rate switch on the TDC controller to the User Baud position.
2) Load the user project into the TDC controller with the desired new baud rate programmed in the configuration & setup.
3) Set your terminal to the new baud rate using the System menu and selecting Terminal settings and then Com Port.
4) **Cycle power** on the unit to establish communications at the new baud rate. The baud rate switch is only read at power up or reset.

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

5.2.4 Creating a new project

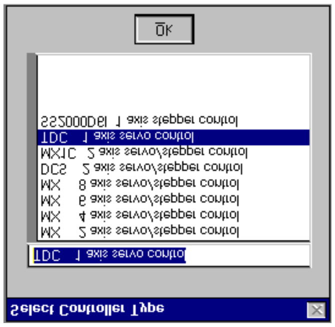
To create a new project either click on the **Create new project** command button on the **TDC opening screen** or the **New** item on the **Project** pull down menu.

New Project Name & Directory Screen



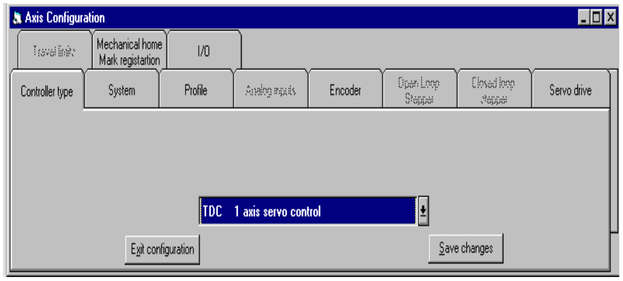
Enter the name of the project **with a .prj extension**. The directory of the project can also be selected at this time. To accept the name and directory click on the **OK** command button.

The controller type can now be selected by clicking on the appropriate check box.



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

Controller type Folder



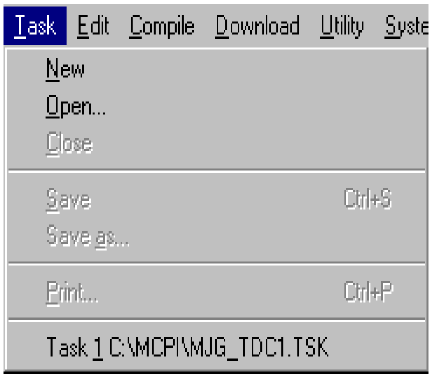
Save each folder that is changed by clicking on the **Save changes** command button. After completing all the changes to the configuration click on the **Exit configuration & setup** command button.

Note: The configuration can be edited later if any changes are required.

5.2.5 - The Task Editor

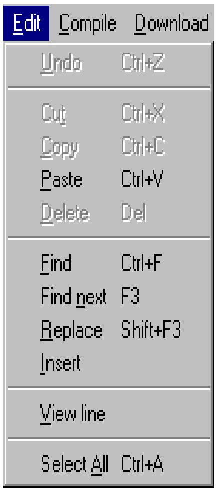
The Project program is created and edited using the Task Editor. To select the project to be edited click on the **Task** menu and either the **New** or **Open** item. The **New** selection allows a new task to be developed. The **Open** selection allows a previously developed task to be edited.

Task Menu Screen



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

Edit Menu



Undo (Ctrl+Z) undoes the latest deletion

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

Copy (Ctrl+C) copies the selected text to the clip board.

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

Delete (Del) deletes the selected text.

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

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

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

Insert Inserts a selected file at the current position.

View line go to selected line number.

Select all (Ctrl+A) selects all text.

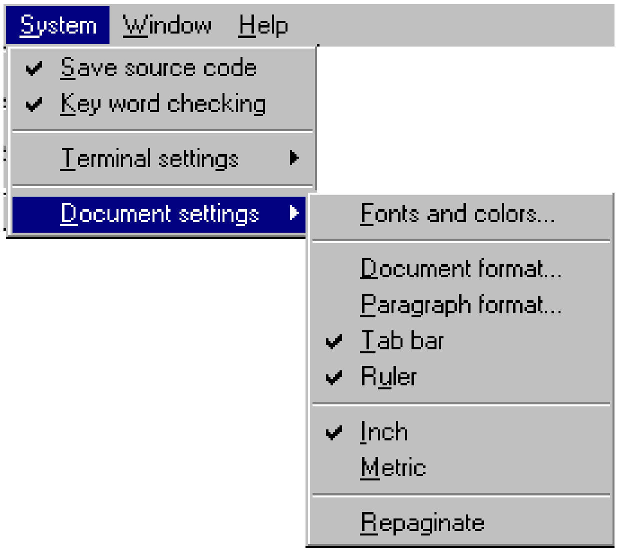
Keyword checking enables or disables Keyword checking. If Enabled it Capitalizes keywords such as program commands and uses the selected colors for keywords and comments.

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

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

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

System Menu



Paragraph format selects the document margins, Alignment, line spacing, Tabulator type and Tab spacing. These settings can be duplicated on the Editor Tool Box.

Tab Bar displays the Tab bar when checked.

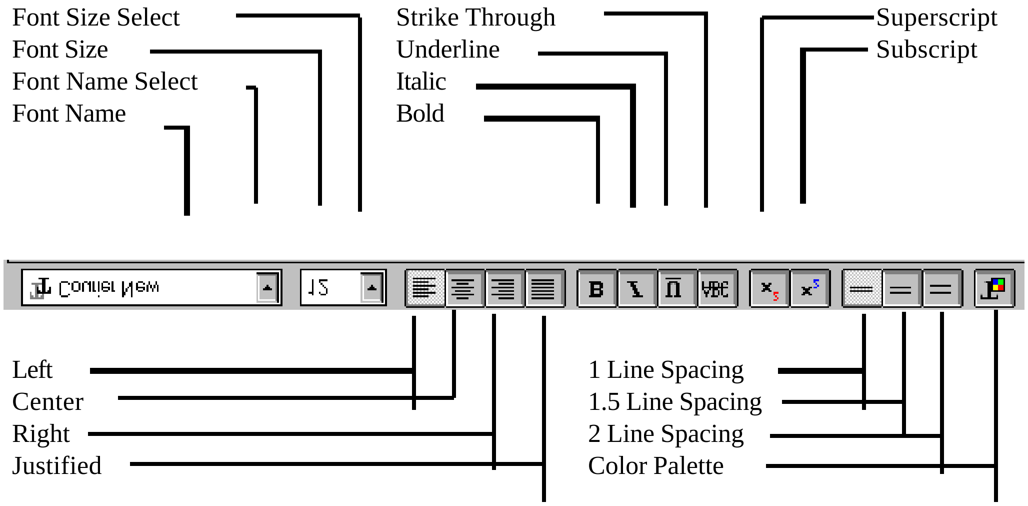
Ruler displays the ruler when checked.

Inch selects the inch ruler when checked.

Metric selects the metric ruler when checked.

Repaginate repaginates the current task.

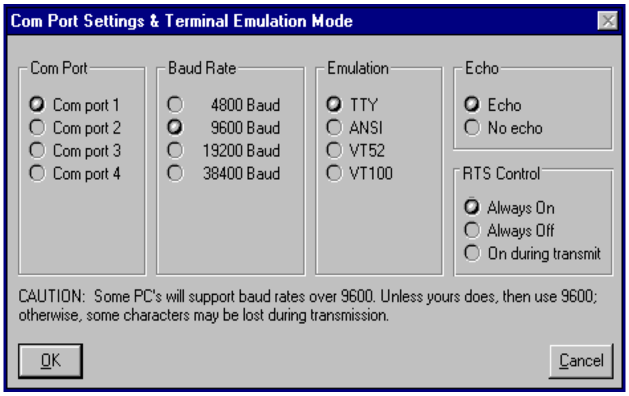
The Editor Tool Box depicted below can be used to modify the text on the Editor Screen. The Font, Type, line spacing and text color can be modified using the Editor Tool Box.



5.2.6 - Terminal Emulation

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 from the Com Port Screen by clicking on one of the circles in each section.

Com Port Screen



Com Port Settings & Terminal Emulation Mode

Com Port
● Com port 1
○ Com port 2
○ Com port 3
○ Com port 4

Baud Rate
○ 4800 Baud
● 9600 Baud
○ 19200 Baud
○ 38400 Baud

Emulation
● TTY
○ ANSI
○ VT52
○ VT100

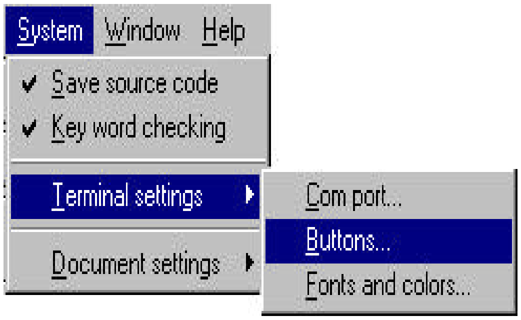
Echo
● Echo
○ No echo

RTS Control
● Always On
○ Always Off
○ On during transmit

CAUTION: Some PC's will support baud rates over 9600. Unless yours does, then use 9600; otherwise, some characters may be lost during transmission.

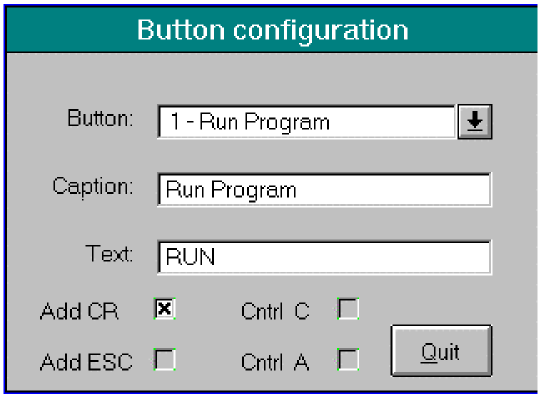
To program the buttons on the Terminal Emulation screen, Click on the **System** Menu and then on the **Terminal settings** item.

System Menu



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

Buttons Screen



Button configuration

Button: 1 - Run Program

Caption: Run Program

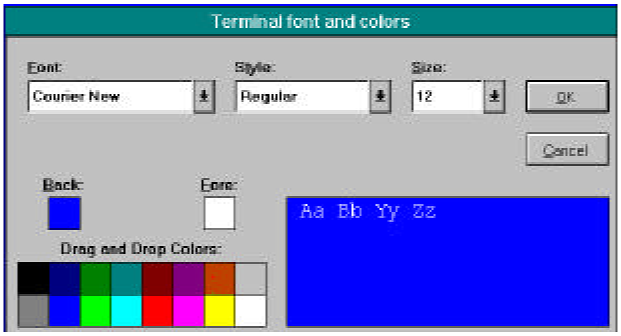
Text: RUN

Add CR ☒ Cntrl C □

Add ESC □ Cntrl A □

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

Fonts and Colors



Terminal font and colors

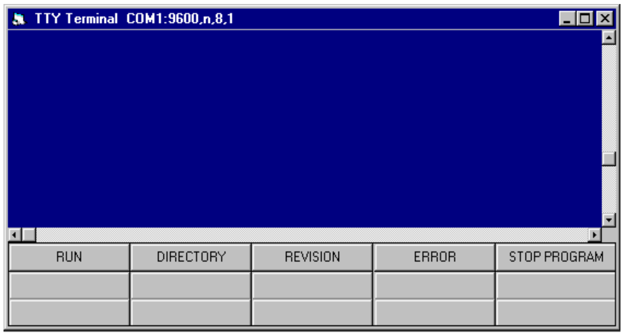
Font: Courier New

Style: Regular

Size: 12

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

Terminal Emulation Screen



TTY Terminal COM1:9600,n,8,1

5.2.7 - Configuration & Setup Folders

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

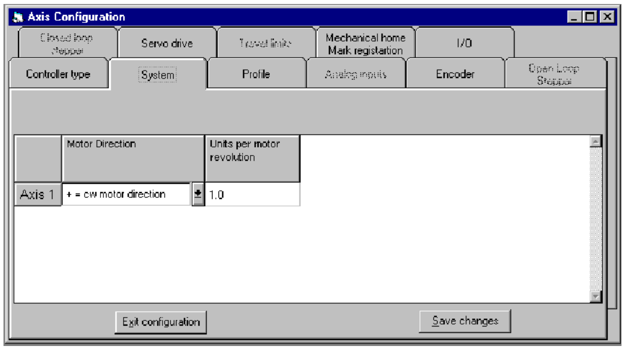
Note: Clicking on the Save changes command button saves the current folder data.

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

Clicking on another folder tab, will allow entry into that folder.

5.2.7.1 - System Folder

This folder specifies the Task assignment for each motor, Drive type, motor direction for a + move and defines the units per motor revolution.



	Motor Direction	Units per motor revolution
Axis 1	+ = cw motor direction	1.0

System Folder TDC Controller

The **Motor direction** sets the motor direction for a + move. The choices are: + for cw motor direction or + for ccw motor direction **as viewed from the non-mounting flange end of the motor.**

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

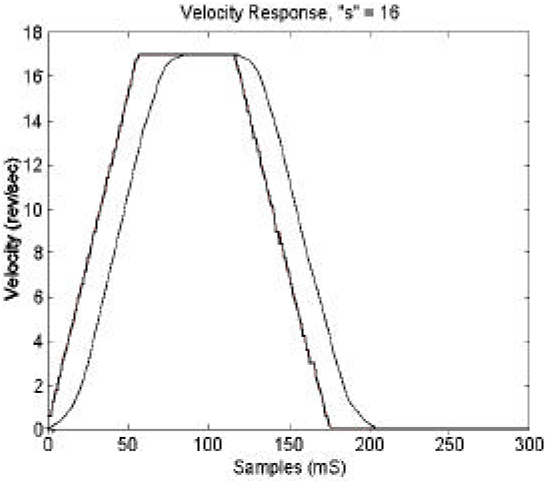
Example:

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

5.2.7.2 - Profile Folder

This folder selects the motion profile, default speed, acceleration rate, deceleration rate, maximum acceleration rate, maximum speed, and minimum time between motions.

Motion Profile determines how the motor's speed changes. Speed changes require a period of accel/decel to increase/decrease the motor's speed. The "Motion Profile" determines how the accel is applied. The MX controller has two choices: trapezoidal or "S" Curve. The TDC controller has 32 choices. In the TDC, 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 "S". The higher the profile setting, the more "S" like the profile. Move times with profile settings 2 - 32 are from 2 to 62 ms longer respectively than those with a setting of 1. The "S-curve" profiles usually results in smoother motion at the expense of longer move times.

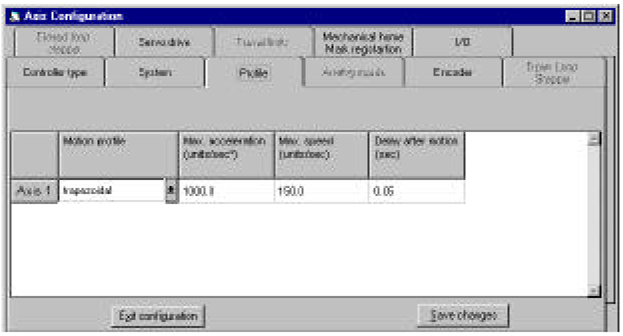


Profile Folder

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

Max Speed sets the maximum allowed target speed in units/sec. Speed, Accel, and Decel 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 parameter sets the minimum time, in seconds, between two moves.



	Motion profile	Max. acceleration (units/sec²)	Max. speed (units/sec)	Delay after motion (sec)
Axis 1	trapezoidal	1000.0	150.0	0.05

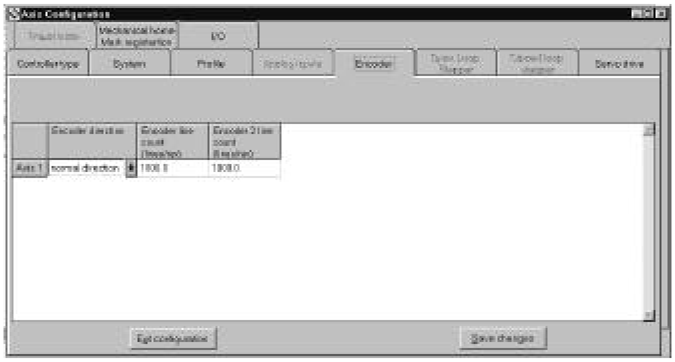
5.2.7.3 - Encoder Folder

This folder allows the Encoder type, Encoder direction, Encoder resolution, and pulse count to be modified.

Encoder direction determines how the encoder rotation direction is interpreted. **If the motor Aruns away@ it is often due to the wrong setting for this parameter.**

Encoder line count defines the encoder resolution in lines. **Encoder 2 line count** defines the external encoder resolution in lines per encoder revolution. For a quadrature encoder the counts per revolution are 4 times the number of 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. Note: In electronic gearing mode the mechanical ratio of the slave motor to the master encoder 2 is specified using the **GEARRATIO** command.

Encoder Folder



	Encoder direction	Encoder line count (lines/rev)	Encoder 2 line count (lines/rev)
Axis 1	normal direction	1000.0	1000.0

5.2.7.4 - Servo Drive Folder

This folder allows the following to be modified: the PID loop gains, acceleration feed forward gain, velocity feed forward gain, integral limit, following error, sample time, and enable/disable integration during motion.

Proportional gain determines the size of the proportional term for a given position error. The units for KP is millivolts per encoder count.

Integral gain determines how fast the integral term grows with non-zero position error. The units for KI are milliseconds. The growth rate is inversely related to the value of KI. For example the integral term grows 5 times faster with KI = 10 than with KI = 50. A special case is KI = 0, which disables the integral action and sets the integral term to zero. When the drive is disabled, the integral term is set to zero. Setting KI only affects the gain for the integral term.

Derivative gain determines the derivative gain and affects the gains for the velocity and feed forward terms. This gain is specified in milliseconds.

Velocity Feed Forward can be used to reduce the position error during motion and is specified in percent. It does not affect system stability. The minimum error occurs with KVFF near 100%. Setting KVFF only affects the gain for the feed forward term.

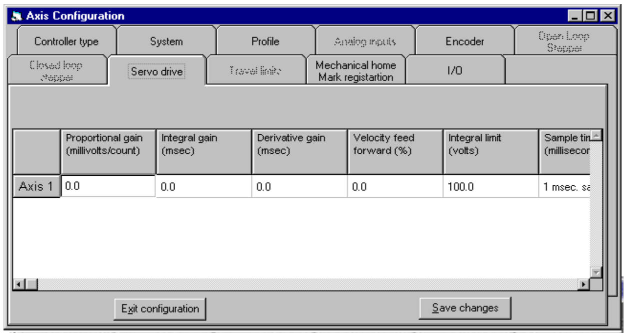
Integral limit limits the contribution of the integral term to the servo loop's output. The limit can be set between 0 and 319 volts inclusive.

Following error defines the maximum error allowed during motion in units. Following error for the TDC must be set in the user program.

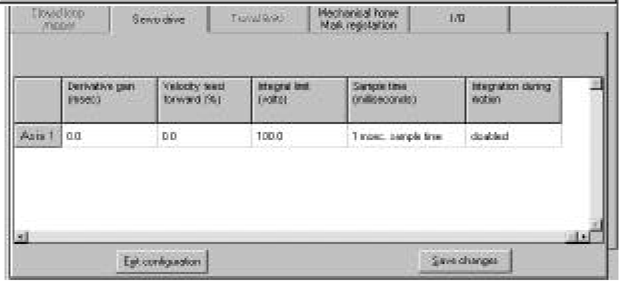
Sample time determines how often the servo loop output is updated. The possible settings are 1 or 2 ms. Use 1 ms. if the encoder count rate is above 1,000,000 counts/sec.

Integration during motion determines whether the integral term contributes to the servo output during motion. If enabled, the integral term is always active. If disabled, the integral term is active only when motion is not commanded.

Servo Drive Folder



	Proportional gain (millivolts/count)	Integral gain (msec)	Derivative gain (msec)	Velocity feed forward (%)	Integral limit (volts)	Sample tin (millisecor
Axis 1	0.0	0.0	0.0	0.0	100.0	1 msec. sa



	Derivative gain (msec)	Velocity feed forward (%)	Integral limit (volts)	Sample time (milliseconds)	Integration during motion
Axis 1	0.0	0.0	100.0	1 msec. sample time	disabled

Servo Drive Folder (Cont.)

5.2.7.5 - Mechanical Home & Mark Registration Folder

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

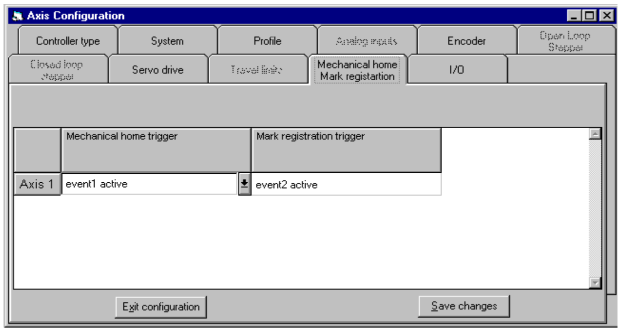
Mechanical Home trigger & **Mark Registration trigger** specifies the trigger for the cycle. There are two trigger inputs EVENT 1 and EVENT 2 which can be used as a trigger.

The trigger combinations for mechanical home and mark registration are:

Event 1 active (rising edge of Event 1 input)
Event 1 inactive (falling edge of Event 1 input)
Event 1 active **&** encoder marker (Event1 =1 and enc. marker =1)
Event 1 inactive **&** encoder marker (Event1 =0 and enc. marker =1)
Encoder marker active (rising edge of enc. marker input)
Encoder marker inactive (falling edge of enc. marker input)
Event 2 active (rising edge of Event 2 input)
Event 2 inactive (falling edge of Event 2 input)

In the above selections the **&** signifies that BOTH conditions must exist for the trigger to occur.

Mechanical Home & Mark Registration Folder



	Mechanical home trigger	Mark registration trigger
Axis 1	event1 active	event2 active

5.2.7.6 - I/O Folder

This folder is used by the TDC controller only.

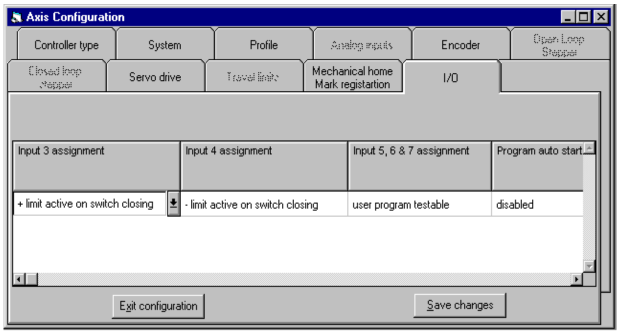
Inputs 3-7 can be assigned as general purpose inputs or as dedicated functions. Also, the auto program start can be enabled or disabled, and the host baud rate for the User position on the TDC can be programmed.

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

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

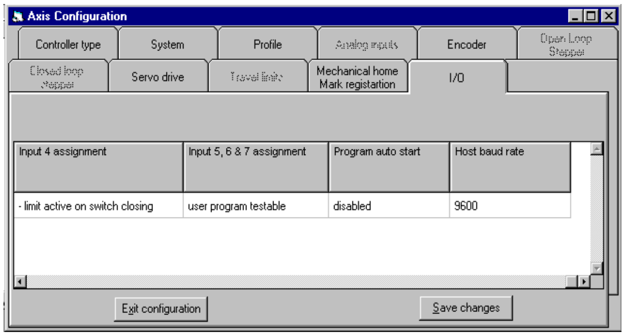
Run - Program execution is started using the run input or the host "RUN" command. At power up an active run input will start program execution unless the clear input is inactive. Following the initial power up, an inactive to active transition on the run input does the following:
If the Feedhold state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

I/O Folder



Input 3 assignment	Input 4 assignment	Input 5, 6 & 7 assignment	Program auto start
+ limit active on switch closing	- limit active on switch closing	user program testable	disabled

I/O Folder (Cont.)



Input 4 assignment	Input 5, 6 & 7 assignment	Program auto start	Host baud rate
- limit active on switch closing	user program testable	disabled	9600

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

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

The **Program auto start** feature may be enabled for a power-on condition or reset command. The user may enable this feature to allow the program to start executing upon a power up or reset condition automatically. The **Host baud rate** may be set to 9600, 19200, or 38400. Once the corresponding Project is downloaded into the TDC controller, this rate will be used for serial communications to the Host (usually a PC) if and only if the Baud Rate Switch on the TDC unit is set to the User Baud position. After downloading, the new baud rate will take effect upon power up or reset. If the baud rate switch is in the 9600 position, all communications will occur at 9600 Baud.

5.2.8 - Preparing User Project for Execution

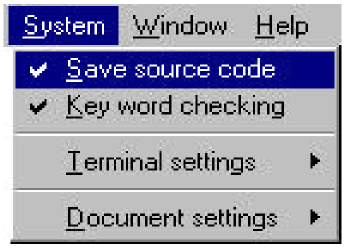
In order to execute a project program it must first be compiled and Downloaded to the controller. The project source code can be recovered from the controller as well.

5.2.8.1 - Project Source code

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

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

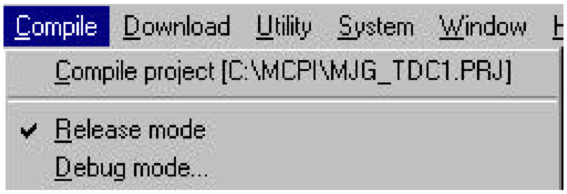
Project Source Selection



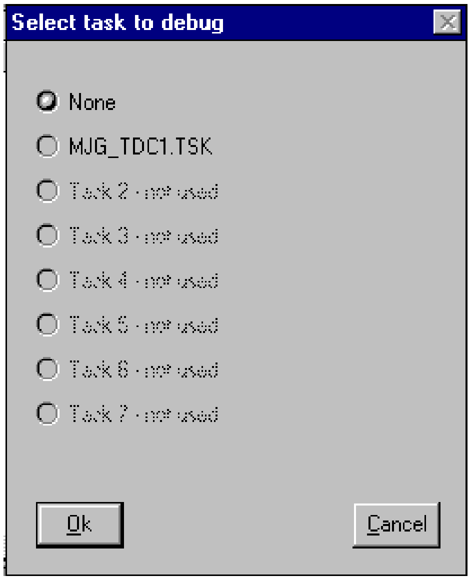
5.2.8.2 - Setting Project Debugging

To select the task to be debugged click on the **Compile** menu and then on the **Debug mode** item. Then select the task by clicking on the task name. The project must now be compiled and downloaded before task debugging can begin. To cancel the debugging mode selection click on the **Compile** menu and then the **Release mode** item. To complete this cancellation the project must now be re-compiled and then downloaded.

Compile Menu



Debug Task Selection



5.2.8.3 - Compiling a Project

A project can be compiled by clicking on the **Compile** Command button or on the **Compile** menu and then the **Compile project** item.

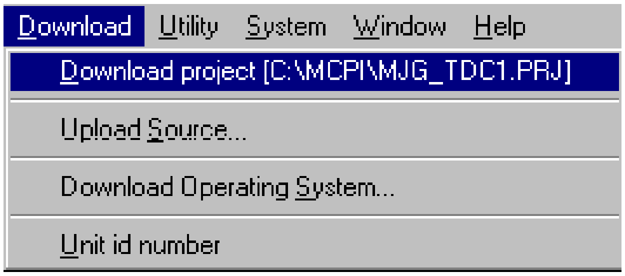
5.2.8.4 - Downloading a Project

A project can be downloaded with or without its source code by clicking on the **Download** command button or clicking on the **Download** menu and then the **Download project** item.

5.2.8.5 - Uploading Source Code

A projects source code can be uploaded from the controller to the PC by selecting the **Upload Source** item from the **Download menu**. A project can be uploaded from the controller ONLY if it had previously been saved in the controller. See section 5.2.8.1.

Download & Upload Project


Download
Utility
System
Window
Help
Download project [C:\MCPI\MJG_TDC1.PRJ]
Upload Source...
Download Operating System...
Unit id number

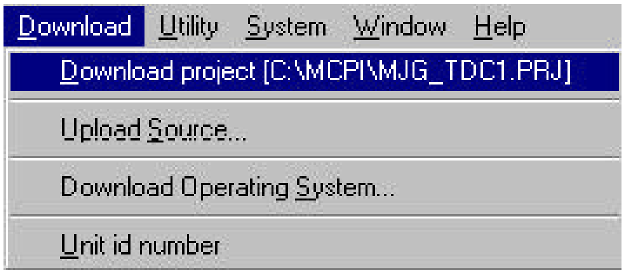
5.2.9 - Downloading an Operating System

New operating system software which runs the controller can be downloaded by clicking on the **Download menu** and then the **Download Operating System** item.

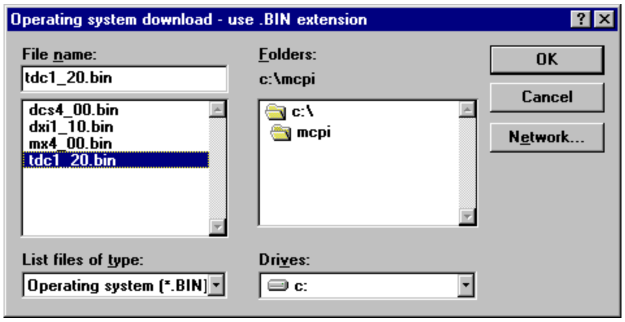
The operating system file, with extension **.bin**, can be selected by clicking on the desired file name. To start the operating system download procedure click on the **OK** command button.

Note: The file names for the different controllers start with the following letters: **mx** for the MX2000 controller, **dcs** for the DCS controller and **tdc** for the TDC controller.

Download Operating System Selection


Download
Utility
System
Window
Help
Download project [C:\MCPI\MJG_TDC1.PRJ]
Upload Source...
Download Operating System...
Unit id number

Operating System File Selection



File name: tdc1_20.bin

Folders: c:\mcpi

List files of type: Operating system (*.BIN)

Drives: c:

5.2.10 - Other Menus

The MCPI menus are pull down menus. Clicking on a menu shows an itemized list of operations allowed for that menu. The menus are: Project, Task, Edit, Compile, Download, Utility, System, Window and Help.

5.2.10.1 - Project Menu

This menu allows you to create a new project, open an existing project, save a current project, add or remove a task from a project, open the configuration & setup environment, print a current project, or exit the MX2000 - TDC programming environment.

New is used to create a new project.

Open is used to open up an existing Project.

Save is used to save the current project.

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

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

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

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

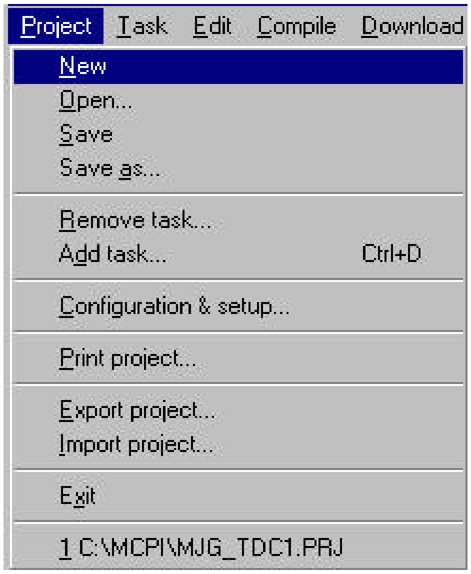
Print project is used to print a current project=s information.

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

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

Exit is used to exit the MCPI programming Environment.

Project Menu Items



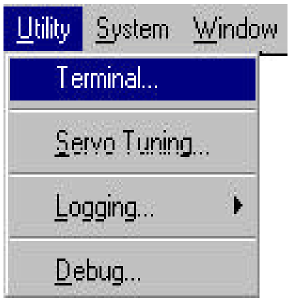
5.2.10.2 - Utility Menu

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

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

Servo Tuning allows the tuning of a servo system.

Debug starts program task debugging.



Utility Menu Items

5.2.10.3 - Window Menu

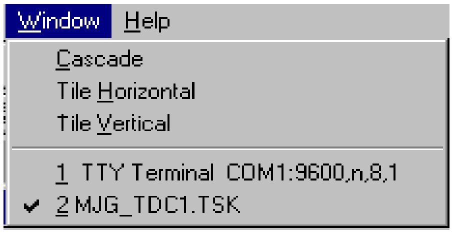
This menu selects the windows format for the open windows.

Cascade cascades the open windows.

Tile Horizontal tiles the open windows Horizontally.

Tile Vertical tiles the open windows Vertically.

Window Menu Items



5.2.10.4 - Help Menu

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

Contents list the help topics.

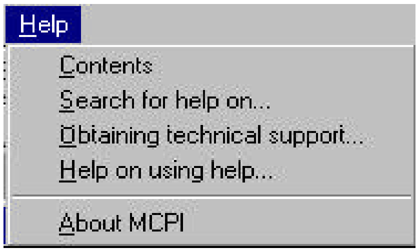
Search for help on lists the help items and descriptions.

Obtaining technical supportprovides application assistance telephone numbers.

Help on using helpprovides help on how to use Help.

About MCPI provides the MCPI version number.

Help Menu Items



5.2.11 - Project Command Buttons

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

Configuration enters the configuration & setup environment.

Compile project compiles a current project.

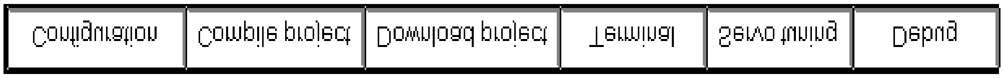
Download project downloads a current project.

Terminal enters the Terminal emulation environment.

Servo tuning enters the Servo tuning environment.

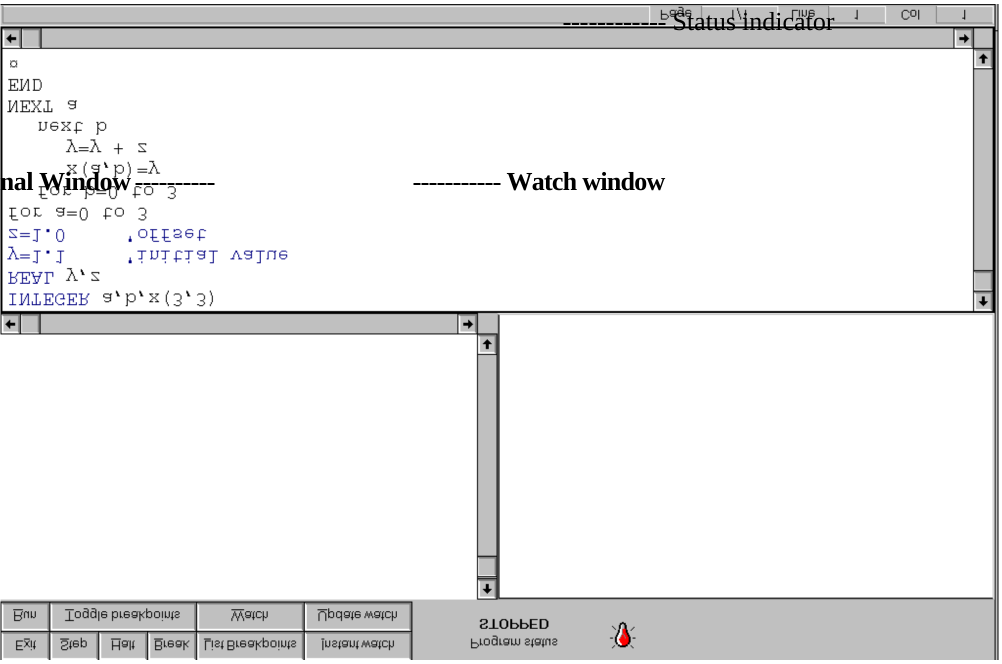
Debug enters the Program Debugger Environment.

Project Command Buttons



5.2.12 - DEBUG Environment

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



```
INTEGER a,b,x(3,3)
REAL y,z
y=1.1      'initial value
z=1.0      'offset
for a=0 to 3
   for b=0 to 3
      x(a,b)=y
      y=y + z
   next b
NEXT a
END
o
```

Command buttons --

------------ Status indicator

Terminal Window ----------

----------- Watch window

Program Window ---

5.2.12.1 - Debug program execution

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

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

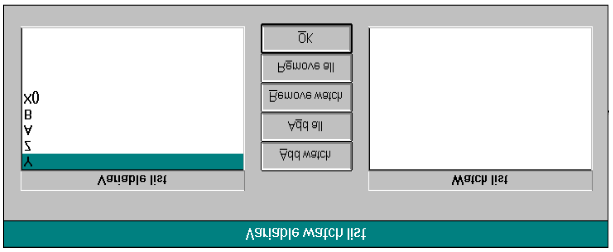
5.2.12.2 - Breakpoint Setting/Clearing

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

5.2.12.3 - Watch variables

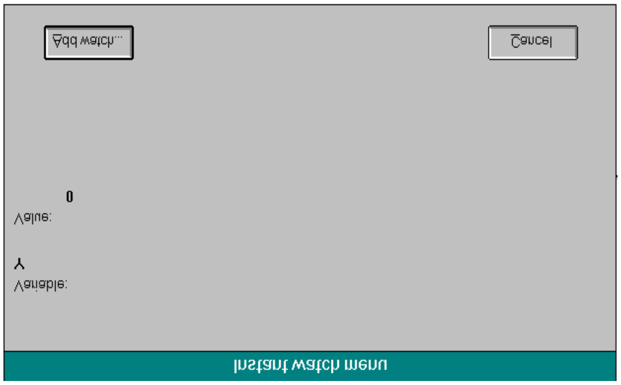
Variable watch allows the programmer to view the values of selected variables. To add or remove a watch variable from the watch window click on the **Watch** command button.

Watch List Screen



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

Instant Watch Screen



Variable:
Y
Value:
0

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

5.2.12.4 - Terminal Window

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

5.2.12.5 - Exit Debug Environment

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

5.2.13 - Servo Control

A servo is a closed loop system. The loop is closed by taking a measurement of the actual output (usually a position or velocity) and comparing it to the desired command or reference input. An error signal is generated by subtracting the output signal from the reference. The error signal tells the controller how far away the output is from the desired position. Then, a control law (algorithm) modifies this error signal to provide an output voltage to drive the servo amplifier.

The controller uses a modified form of the classic PID (Proportional, Integral, Derivative) control law with velocity feed forward. The Commanded position is compared to the Encoder position and a Position error is generated. A control algorithm modifies this error to provide an output voltage to drive the Servo amplifier. The PID control loop uses or derives the following parameters or commands:

Configuration only

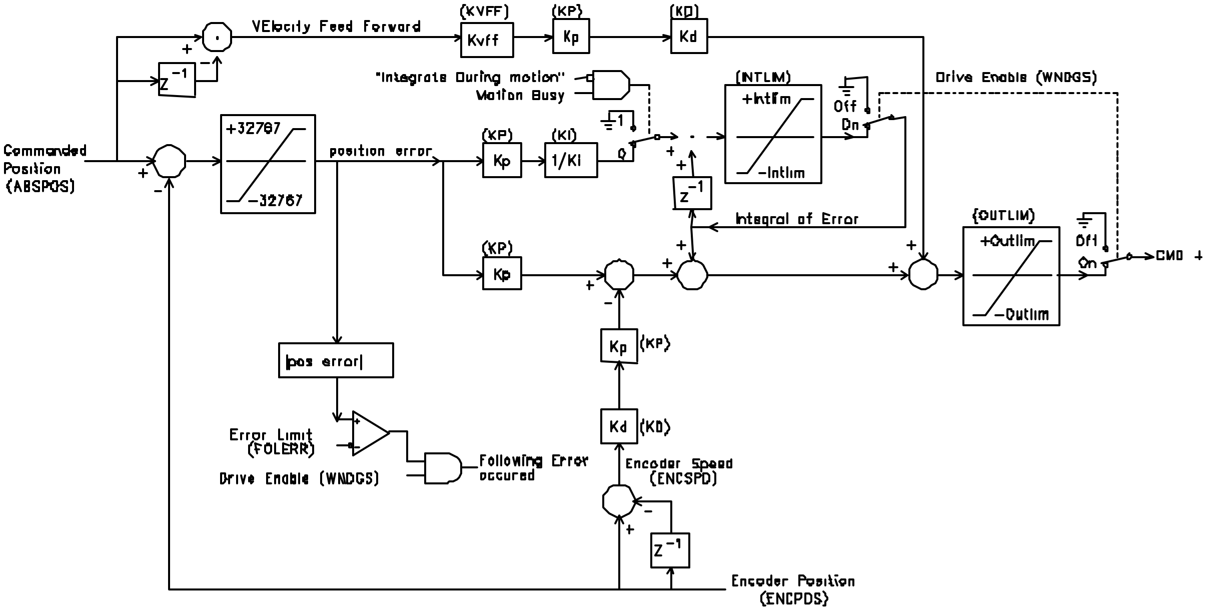
Integration during motion
Sample time
Encoder line count
Encoder direction

Configuration & Program

Kp	(Proportional gain)
Ki	(Integral gain)
Kd	(Derivative gain)
Kvff	(velocity feed forward gain)

Program only

OUTLIM
WNDGS
ENCPOS
ENCSPD
INTLIM (integral limit voltage)
ABSPOS
FOLERR



TDC BLOCK DIAGRAM

5.2.14 - Servo Tuning

Tuning is a process of determining the PID and feed forward gains to obtain the desired system response. Typical performance indicators like: overshoot, response time, stiffness, settling time, bandwidth and damping can all be used to measure how well the system is tuned. Tuning a gain to improve its performance characteristic may cause another characteristic to get worse.

Before attempting servo tuning the following must be done: Modify the configuration & setup and Compile and download the user project to the controller**.**

<u>System folder</u>

Units per motor revolution set to the desired value.

<u>Encoder folder</u>

Encoder line count set to the encoder line count of the motor encoder.

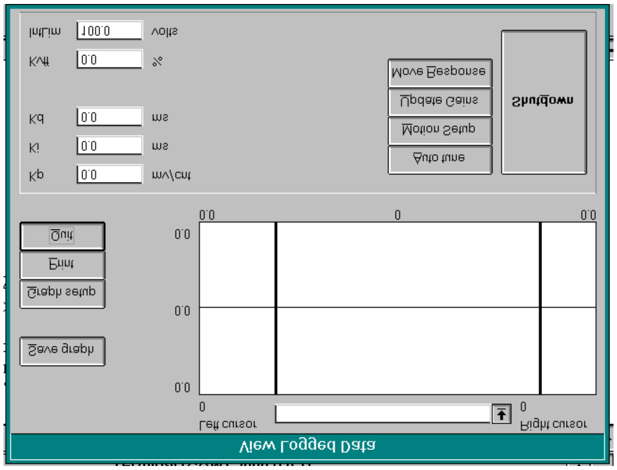
<u>Servo drive folder</u>

Integral Limit set to the desired integral voltage limit or use the default value.

Sample time of the servo drive set to one msec or two msecs.

Integration during motion set to enable or disable.

Note: A program does not have to be written in the task in order to tune the servo motor



Intlim 1000.0 volts
Kvff 0.0 %
Kd 0.0 ms
Ki 0.0 ms
Kp 0.0 mv/cnt

To access the servo tuning environment, click on the **Servo Tuning** command button.

<u>Command Buttons</u>

Save Graph saves the current displayed graph.

Graph setup allows for the selection of color and style for each logged item.

Print prints the current graph.

Quit exits the Servo Tuning environment.

Auto tune enters servo auto tune environment.

Motion Setup allows setting of motion parameters.

Update Gains sends the current servo gains to the controller.

Move Response creates a logged move cycle.

Shutdown disables the servo output voltage.

Drop List

Display Drop list selects the logged item to be displayed.

Text Boxes

Kp displays the current value of the proportional gain.

Ki displays the current value of integral gain.

Kd displays the current value of derivative gain.

Kvff displays the current value of the velocity feed forward gain.

IntLim displays the current value of integral limit.

5.2.14.1 - Auto Tuning

Before a servo can run properly, the servo gains Kp, Ki, Kd, and Kvff must be set up to yield the appropriate move response. The controller has the ability to automatically set the servo gains using an automatic tuning procedure The auto tuning environment is accessed by clicking on the **Auto tune** command button.

There are several steps to the automatic servo tuning process:

1) Measure System Gain.

The system gain is a measure of the overall responsiveness of the system. Higher inertia and/or lower torque yields lower system gain. Lower inertia and/or higher torque yields higher system gain. The system gain number is used when the software calculates the servo gains. In order for the motor to track a given profile response, the lower the system gain, the higher the calculated overall controller gain will be to compensate and vice versa.

Clicking **Measure System Gain** instructs the controller to provide a “bump” of torque to the motor. Three parameters, **Output, Speed**, and **Distance Limit** are used to measure system gain.

! Caution

When the Measure System Gain button is clicked, the motor will move quickly and abruptly for a short distance.

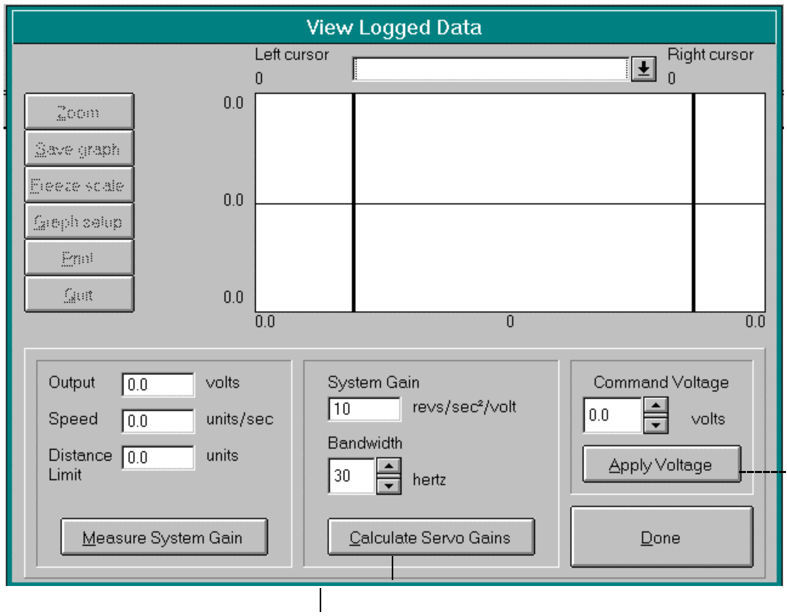
The **Output** text box is used to select the amount of voltage that the controller will use to bump the motor. The range of the Output is 0 to 10 volts where 10 volts represents peak torque. Typically the default parameter of 2 volts is adequate, although some large inertia systems may require the Output be set to 3 or 4 volts.

The **Speed** text box is used to select the target velocity for the gain measurement. During gain measurement, the output torque will be applied to the motor until the speed set here is reached. The default speed is usually sufficient.

The value in the **Distance Limit** text box limits the distance that the motor will turn during the gain measurement. If the **distance limit** is reached before the motor reaches the **speed** indicated, or if the speed cannot be reached with the voltage entered, an error message will appear. If an error appears, try increasing the distance limit or raising the voltage output slightly.

Generally, the default parameters for these three parameters should be used during the gain measurement.

If the gain measurement is unsuccessful, verify that the motor moves properly with a constant torque command applied. This may be done by using the **Apply Voltage** button in the manual motion area of the auto tune screen. Clicking the **Apply Voltage** button will output a constant torque to the motor proportional to the command voltage. Start with zero and click on the up or down arrows to apply positive or negative torque respectively. Unless the system has high friction, the motor should begin to move with less than one volt applied. Check that the motor torque is smooth and continuous in both directions by applying small amounts of positive and negative voltage.



Left cursor 0

Right cursor 0

Output 0.0 volts

Speed 0.0 units/sec

Distance Limit 0.0 units

System Gain 10 revs/sec²/volt

Bandwidth 30 hertz

Command Voltage 0.0 volts

-------manual motion

gain measurement-------

gain calculation

2) Set the Bandwidth

The system bandwidth is essentially the maximum frequency of excitation the system will respond to. Generally, higher bandwidth systems are “stiffer” or “tighter”. Lower bandwidth systems are “soft” or “sluggish”. Generally bandwidths range from 10 to 60 Hz (cycles per second). The auto tuning procedure uses the bandwidth setting along with the measured system gain to calculate the appropriate servo gains for the system. The default bandwidth of 30 Hz is usually a good starting point, although sometimes the it must be lowered to achieve a stable system, or raised to achieve a fast enough response.

3) Calculate Servo Gains

Clicking the Calculate Servo Gains button will use the bandwidth and measured system gain to calculate the Kp, Ki, Kd, Intlim (and Kaff if applicable) parameters. These fields will be updated after the calculation is complete. Click **Done** to return to the main servo tuning screen.

4) Update Gains

Clicking Update Gains updates the gains to the controller immediately. **Caution!** Updating the gains may change the dynamics of the system such that it becomes unstable and oscillatory. If a loud buzzing or vibration occurs after updating gains, the **Shutdown** button should be clicked. It’s also possible that a fault will occur if the oscillation overtaxes the servo drive. In this case you will have to enter the terminal screen and clear the error by typing ERR or ERRM. If necessary, re-enter the auto tuning screen and **lower the bandwidth** and re-calculate the servo gains. Now update the gains again. Repeat this process until the system is stable and will smoothly resist loading in both directions.

5) Motion Setup

Clicking Motion Setup allows test motion profile parameters to be entered so that the proper motion response may be verified. **Accel, Decel, Speed** and **Move Distance** parameters describe the move that the motor will try to follow during the test. Units of measure are used form the system folder located in the Configuration screen. The **display time** is adjustable so that shorter or longer moves may be fully displayed. Click **Done** to leave the motion setup screen.

6) Move Response

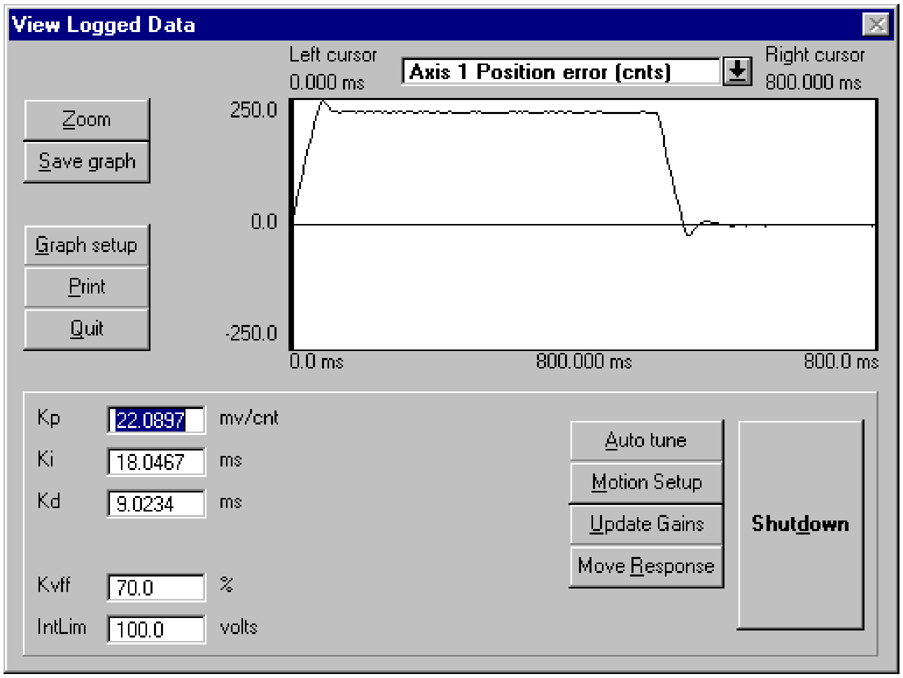
Once the motion setup is complete, the system is ready to attempt to execute the move. Clicking **Move Response** will command the motor to execute the move profile as described in the motion setup screen. The controller will log the response of the motor and display the results on the screen graphically. The position error, torque command, encoder velocity, etc. may be viewed by clicking on the drop down list at the top of the window. The displayed graph of the position error is the error based on quadrature signal feedback from the encoder (for example there are 4000 counts or pulses per revolution on a 1000 line encoder).The response may be observed to verify proper performance for the programmed profile. If the response is acceptable, Quit the servo screen and Save the configuration. You will now have to Compile and Download the project for the new servo information to permanently take effect.

7) Integrate During Motion

This feature allows you to select whether the integration gain is used during the profile motion. Enabling the integrator during motion will reduce your position error at speed, but may cause some unacceptable overshoot in the response. Some controllers allow you to set this parameter in the servo tuning screen, while others require that you change it in the Servo folder in the program configuration (be sure to compile and download the project each time you change the configuration or the change will not take effect).

The figure below shows a stable response with and without integration during motion enabled.

Note, your screen may look slightly different, but the response display will be identical.



View Logged Data

Left cursor 0.000 ms

Axis 1 Position error (cnts)

Right cursor 800.000 ms

Kp 22.0897 mv/cnt

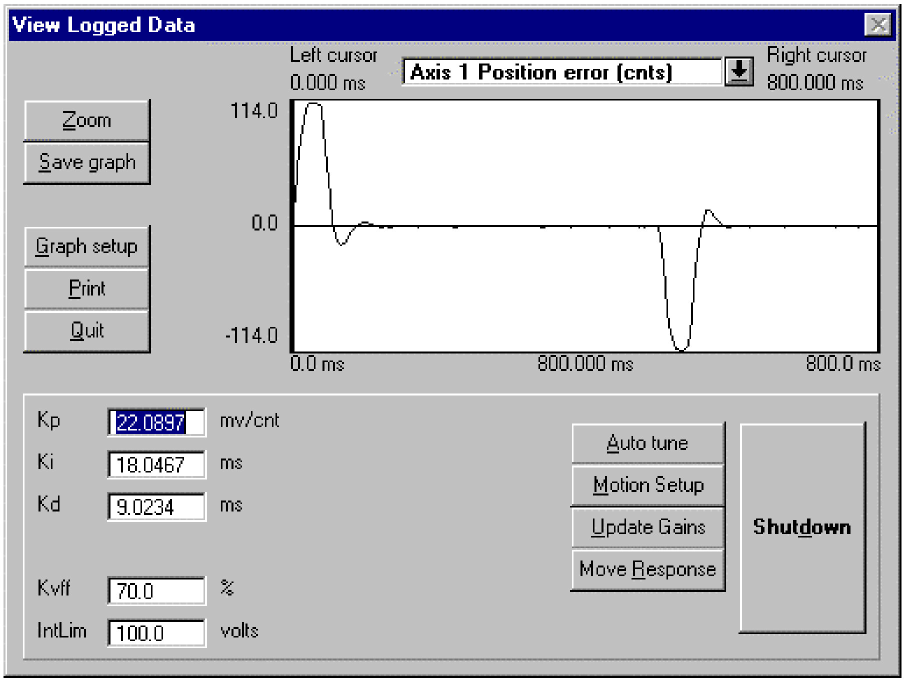
Ki 18.0467 ms

Kd 9.0234 ms

Kvff 70.0 %

IntLim 100.0 volts

Stable response with integration during motion disabled.



View Logged Data

Left cursor 0.000 ms

Axis 1 Position error (cnts)

Right cursor 800.000 ms

Kp 22.0897 mv/cnt

Ki 18.0467 ms

Kd 9.0234 ms

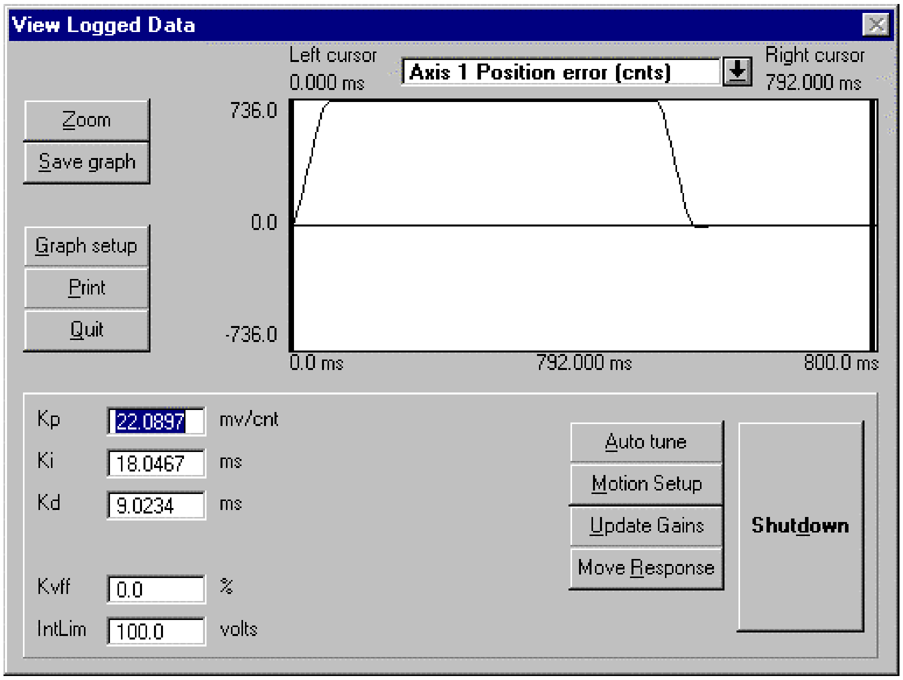
Kvff 70.0 %

IntLim 100.0 volts

Stable response with integration during motion enabled.

8) Velocity Feedforward

This term reduces the error during motion. It should typically be set between 50% and 100%. The figures below show a response with Kvff set to 0%, 50% and 100%. In all three cases the integration during motion was disabled.



Left cursor 0.000 ms

Axis 1 Position error (cnts)

Right cursor 792.000 ms

Kp 22.0897 mv/cnt

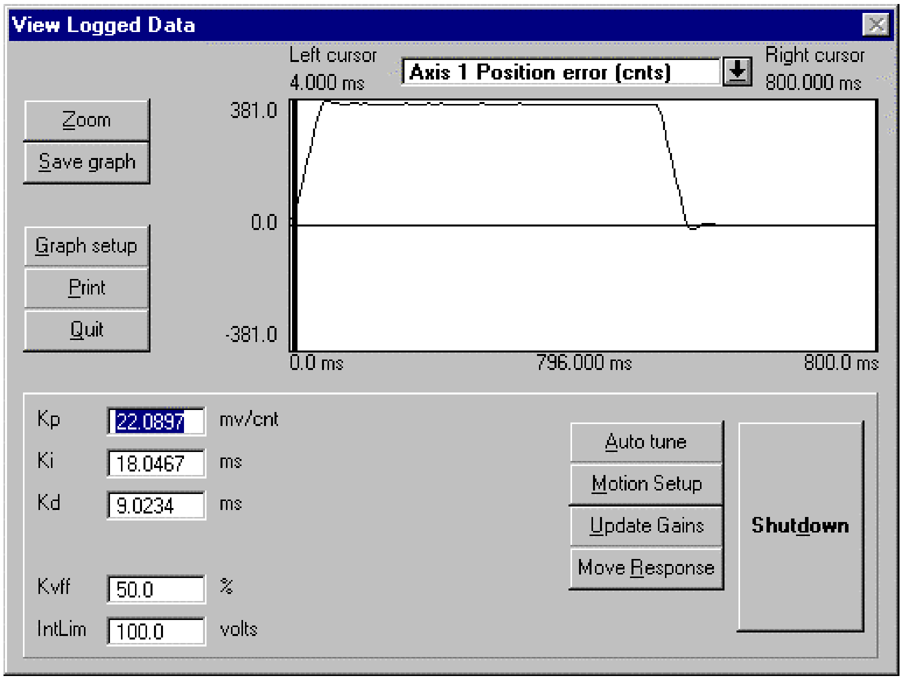
Ki 18.0467 ms

Kd 9.0234 ms

Kvff 0.0 %

IntLim 100.0 volts

Response with Kvff = 0%.



Left cursor 4.000 ms

Axis 1 Position error (cnts)

Right cursor 800.000 ms

Kp 22.0897 mv/cnt

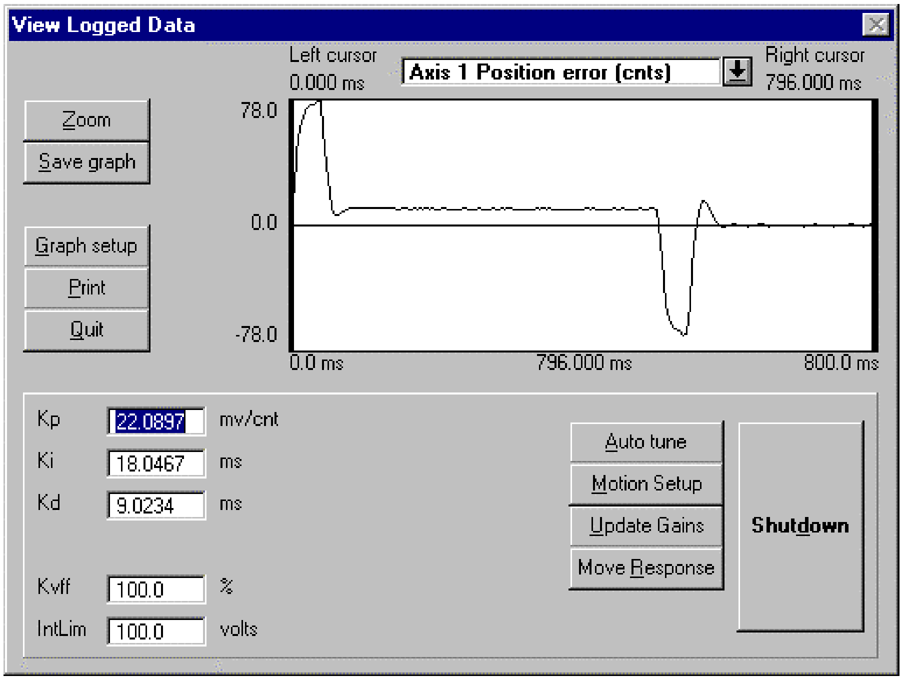
Ki 18.0467 ms

Kd 9.0234 ms

Kvff 50.0 %

IntLim 100.0 volts

Same profile as above with Kvff = 50%. Note reduction in error.



Left cursor 0.000 ms

Axis 1 Position error (cnts)

Right cursor 796.000 ms

Kp 22.0897 mv/cnt

Ki 18.0467 ms

Kd 9.0234 ms

Kvff 100.0 %

IntLim 100.0 volts

Same profile as above with Kvff = 100%. Note further reduction in error.

5.2.14.2 - Manual Tuning Adjustment

Most applications work acceptably using the results of the auto tuning procedure. However, if the results of auto tuning do not yield a satisfactory move response, the servo gains may be adjusted manually to achieve the required performance. Manual tuning of the servo can be quite involved. Be sure to read this section a few times through before you decide to begin manual adjustments.

The proportional gain (Kp), integral gain (Ki), derivative gain (Kd), velocity feed forward gain (Kvff) and integrator limit voltage (IntLim) can be changed by clicking in the desired text box and typing in the new value. In general, increasing Kp will increase the system responsiveness and lower the error. Increasing Kd will improve system stability. Increasing Ki will decrease the time for integration to have an effect on the position error. If a system is oscillating, it may be useful to set Ki to zero to determine if the integrator is causing the instability. If not, Lowering Kp and/or Kd will stabilize the system. Once stability is achieved, Ki can be set to a high value first, say 50, and then lowered until the desired integration effect is achieved.

PID Parameters

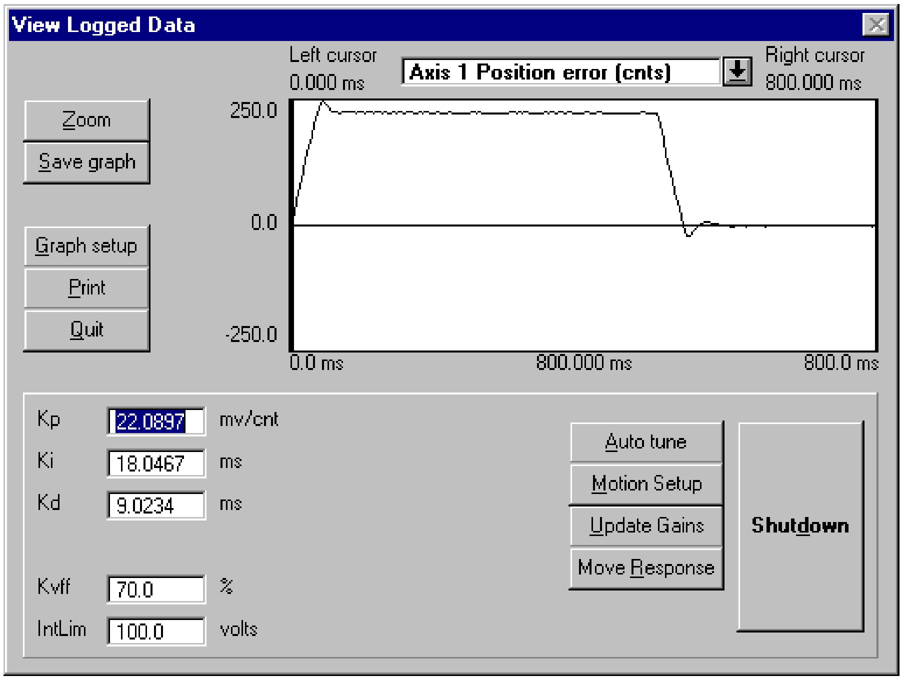
Although manual tuning is possible, all but slight adjustments from the gains calculated with the auto-tuning feature are usually not necessary. **A significant amount of servo experience is usually required to manually tune a system properly. WE RECOMMEND THE AUTO TUNING PROCEDURE BE USED WHENEVER POSSIBLE.**

The single most important rule to remember when adjusting the servo manually is to **gradually change one gain at a time**. There can be interactions between the parameters which will affect the response, and changing more than one gain at a time will certainly lead to confusion.

First let's begin with some definitions along with a description of each parameter and its function. The control loop uses a modified **PID** algorithm to compensate the system response. The servo parameters adjust the controller's output torque command based on **position error**, i.e. the difference between commanded position and encoder position at any given point in time. The **encoder velocity** and **commanded velocity** are also used in some cases. Each parameter contributes to the output torque command in a different way.

Stability or instability:

If the servo system behaves smoothly and without loud buzzing, vibration or oscillation it is said to be **stable.** Conversely, if the system buzzes, vibrates, or oscillates it is said to be **unstable**. The first goal of servo tuning is to achieve a stable system. Once stable the system may be adjusted or "tweaked" to optimize performance. Adjustments should only be made if the response is outright unacceptable. The figures below show a stable and unstable system response.



Left cursor 0.000 ms

Axis 1 Position error (cnts)

Right cursor 800.000 ms

Kp 22.0897 mv/cnt

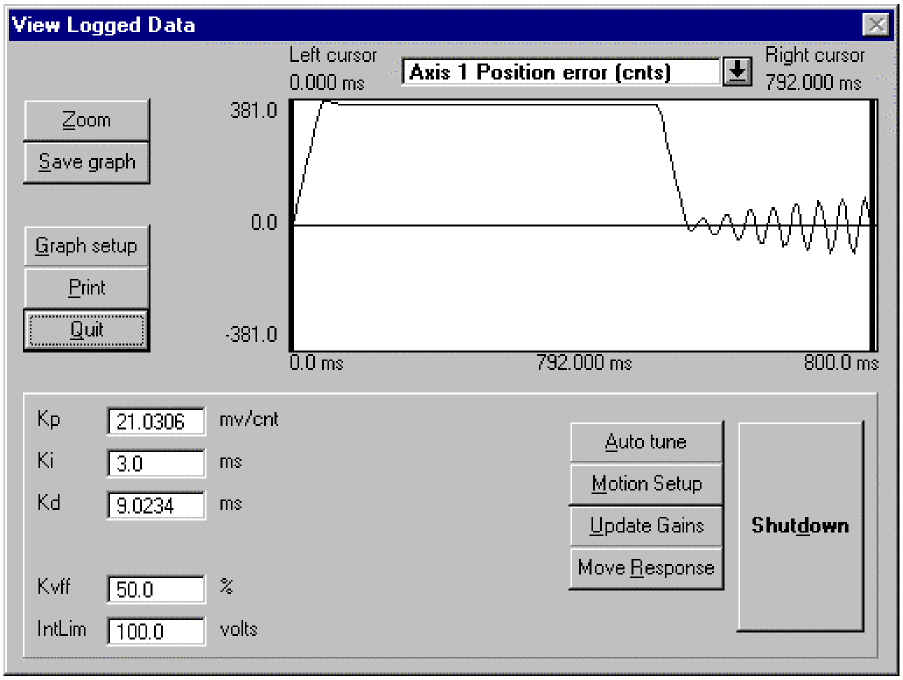
Ki 18.0467 ms

Kd 9.0234 ms

Kvff 70.0 %

IntLim 100.0 volts

Stable response profile.



Left cursor 0.000 ms

Axis 1 Position error (cnts)

Right cursor 792.000 ms

Kp 21.0306 mv/cnt

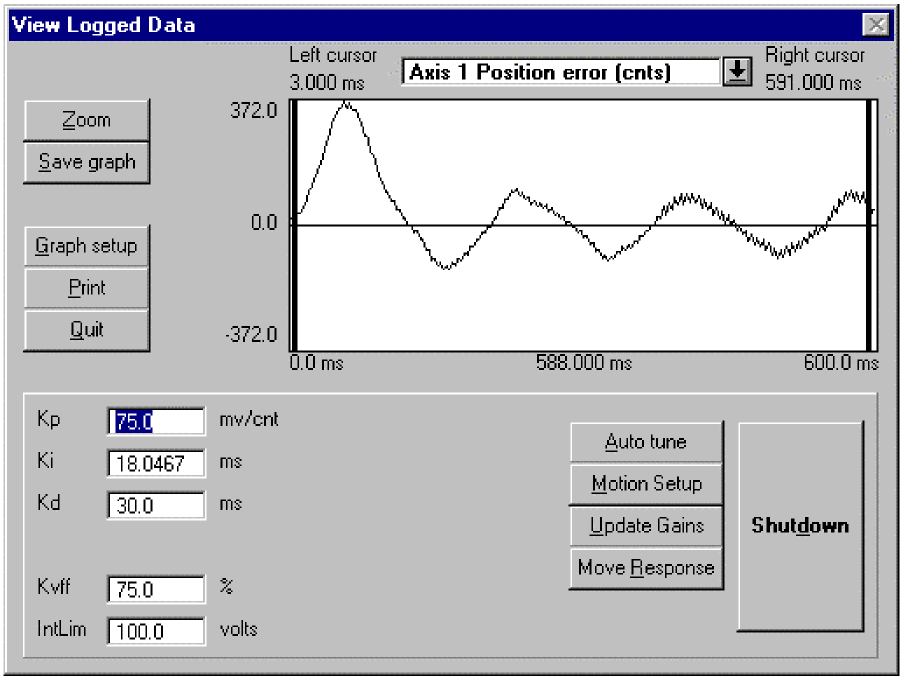
Ki 3.0 ms

Kd 9.0234 ms

Kvff 50.0 %

IntLim 100.0 volts

Unstable response (due to Ki to low).



View Logged Data

Left cursor 3.000 ms

Axis 1 Position error (cnts)

Right cursor 591.000 ms

Kp 75.0 mv/cnt

Ki 18.0467 ms

Kd 30.0 ms

Kvff 75.0 %

IntLim 100.0 volts

Shows unstable response (due to Kp and/or Kd too high). Note "fuzz" from motor "buzzing".

Kp:

Proportional gain. This gain is multiplied by the position error and thus contributes **proportionally** to the output torque. Generally, the higher Kp, the lower the error at any time during the move. However, if Kp is too high, the system can overshoot severely or "buzz" loudly. This type of buzzing instability may be seen as "grass" on the error response curve in the move response screen. In this case, Kp should be lowered. Kd may also be lowered, but to a lesser extent.

Generally the range for Kp is 10 to 150. Kp less than 10 will usually produce a soft or sluggish system. Kp over 175 produces a stiff system, but one that may be approaching instability. Note these are general ranges, not absolute requirements.

Ki:

Integral gain. The reciprocal (1/Ki) of this term is multiplied by the **sum** of the position error over time. The effect of Ki is thus time related, and affects the steady state error. The higher Ki, the longer it will take for the controller to "integrate out" any steady state error. The effect of Ki is seen mostly at constant speed (including standstill). Ki is NOT required for stability, and generally has a de-stabilizing effect on the system, especially if it is too low. If Ki is TOO LOW the system may oscillate slowly and wildly back and forth like a washing machine. Ki is required, though, if the system must achieve a very low steady state error (within a few counts).

The general range for Ki is 10 to 70. Ki less than 10 may lead to wild, low frequency oscillations. If steady state error is not a consideration, Ki may be set to zero. Ki is often disabled during motion to reduce overshoot at the end of the move.

Kd:

Derivative gain. This term is multiplied by the encoder velocity at any point in time. Generally, raising Kd will reduce overshoot in the move response, however, Kd is the term most susceptible to "digital instability". This is where the quantization effects of the digital encoder feedback in conjunction with too high a Kd cause the system to "buzz".

The general range for Kd is 5 to 20. Kd less than 5 usually leads to an unstable system, Kd >20 usually leads to "buzzing".

Kvff:

Feedforward velocity gain. This term is multiplied by the commanded velocity to contribute to the output torque command. It has no effect on general stability, and may be set as high as 100% to reduce position error during motion. Too high a Kvff causes undue motor heating.

Generally, Kvff should be set between 50 and 100.

Kaff:

Some controllers have a Kaff term. This term is multiplied by the commanded acceleration to contribute to the output torque command. This term only takes effect to reduce the error during acceleration and deceleration. Generally Kaff is less than 4. Most applications will run fine with Kaff set at zero.

Adjustment based on auto tune calculation:

It is usually desirable to use the auto tuning gains as a starting point for further adjustment. If the system is unstable at a given bandwidth, the bandwidth may be lowered, and the auto tuning run again. If the move response at this lower bandwidth is unacceptable, the following procedure may be attempted.

Set bandwidth to 25 Hz and calculate gains. Then

1) Update gains and energize system .
2) If the system “buzzes”, lower Kp by 50%, and Kd by 25%.
3) If the system no longer buzzes, check your move response.
4) If the move response over shoots too much, or the system buzzes sometimes, then lower Kp until the buzz goes away and the overshoot is acceptable.
5) Check your move response, and set Kvff between 50-100%. This should reduce the error during the move, and may also improve the overshoot.
6) If the response is well behaved, but sluggish, raise Kp in increments of 2 until acceptable response is achieved. If the system ever “buzzes” Kp must be lowered again.
7) Verify proper response.
8) The system should now be stable and well behaved.

5.2.14.3 - Full Manual Tuning Adjustment

Although it is much more involved, the servo can be tuned “from scratch”. The trick here is to be very **patient and methodical.** Make sure to record each change and its resultant effect on the response. To set up the move, use the Motion Setup as previously described (make sure the motor is sized properly to handle the accelerations and speeds you enter). The bandwidth, measure gain, and system gain are not used in this procedure. Instead you will enter the gains directly into their appropriate fields. Make sure to Update Gains after each adjustment so they take effect. You can use the example response screens at the end of this procedure as a guide.

! Caution

Motor instability can cause severe vibration or sudden movements. Insure that appropriate safety measures such as mechanical limits are employed to prevent dangerous movements of the motor and load.

Manual Tuning Procedure

1) Enter the auto tune screen and select measure system gain.

! Caution

Caution! the motor will move suddenly during this process.

This will verify that the encoder direction is correct for the servo to run properly.

2) If the encoder direction is found to be reversed, then quit the auto tune screen immediately and enter the Configuration. Select the encoder folder and change the encoder direction to the opposite of the present setting. Save the configuration information, compile and download the modified project.

3) Re-enter the servo tuning screen and set Ki, Kvff, and Kaff to zero.

4) Together, set Kp to a low number, say 5, and Kd to a midrange number, say 10.

5) Update the gains and see if the motor is stable by performing a short move using Move Response. Be ready to **shutdown** if the motor oscillates.

6) If the motor is stable and does not vibrate, raise Kp by 2.

7) If not, lower Kp by 1. Repeat until the motor is stable.

8) Once Kp is as high as it will go and still be stable, reduce Kp by **50%** to provide some stability margin.

9) Now try your move response.

10) If the move is stable but overshoots severely, lower Kp slightly. Slight overshoot is o.k. at this point.

11) Continue lowering Kp until the overshoot is close to acceptable.

12) Now we can try to reduce the error during the motion.

13) Set Kvff to 50 and check the response.

14) If the error is not acceptable increase Kvff by 10 and check the response, repeat until the response is acceptable.

15) Now let’s try to use Ki to reduce the error at rest.

16) Set Ki to a high number, say 75 and check the move response.

17) If the response is smooth takes a long time to settle at the end, then decrease Ki by 10. If the motor goes unstable, raise Ki back up again.

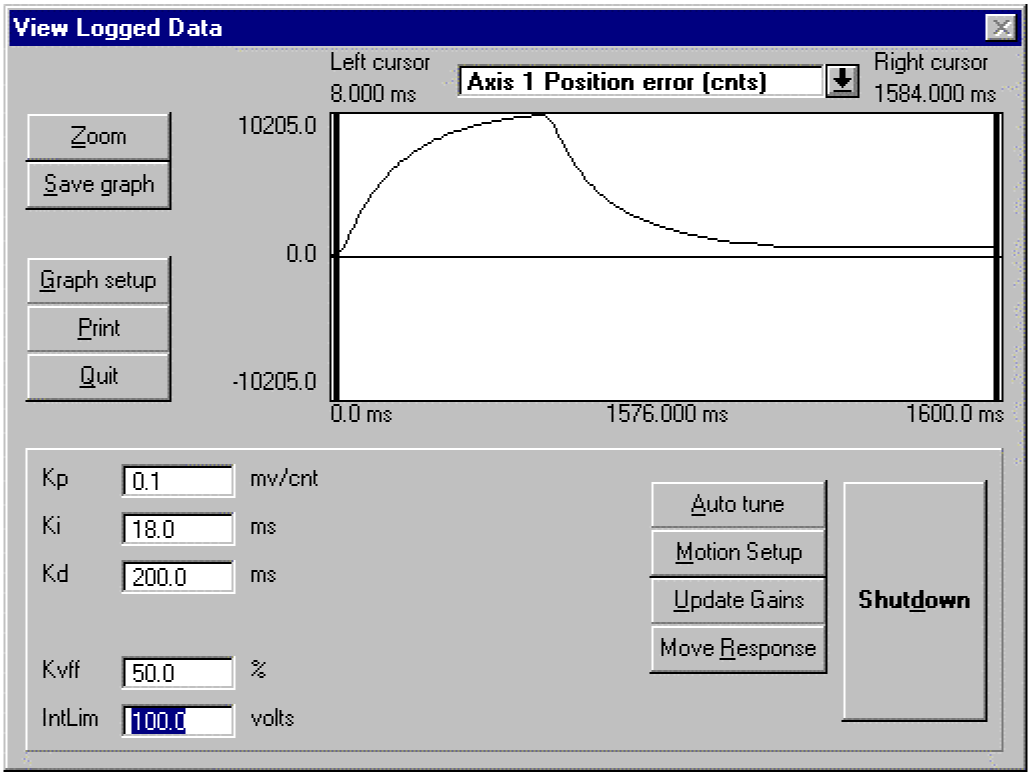
18) Verify the proper response to your profiles.

19) If the response still exhibits oscillation or overshoot, you may need to dampen the system response by raising Kd and repeating the process from step number 5. See the effect of lower Kp and higher Kd in the response graphs below.

20) If the motor will not respond as required, check the torque command response to verify that the controller is not saturating at 10 volts during accel/decel. This would indicate too high an acceleration for this motor and load. Lower the accel or decrease the load inertia.

21) **THAT’S IT!**

The following screens show examples of tuning responses. Each has a description of what caused the response shown.



Left cursor 8.000 ms

Axis 1 Position error (cnts)

Right cursor 1584.000 ms

Kp 0.1 mv/cnt

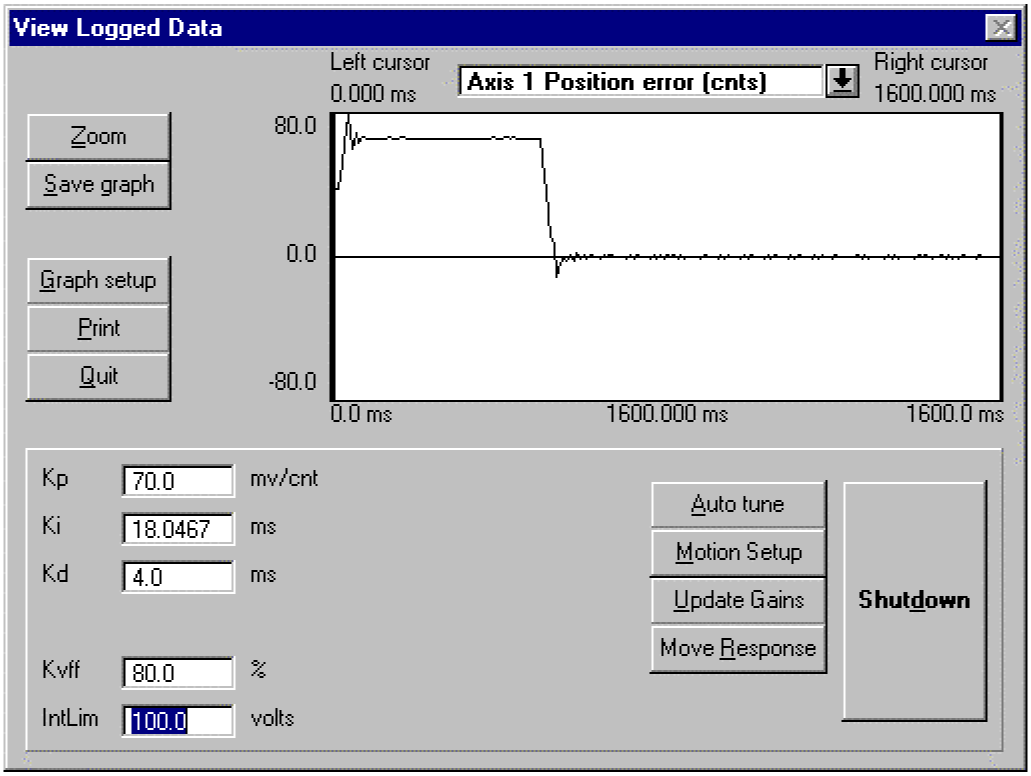
Ki 18.0 ms

Kd 200.0 ms

Kvff 50.0 %

IntLim 100.0 volts

"Sluggish" response due to low Kp and high Kd.



Left cursor 0.000 ms

Axis 1 Position error (cnts)

Right cursor 1600.000 ms

Kp 70.0 mv/cnt

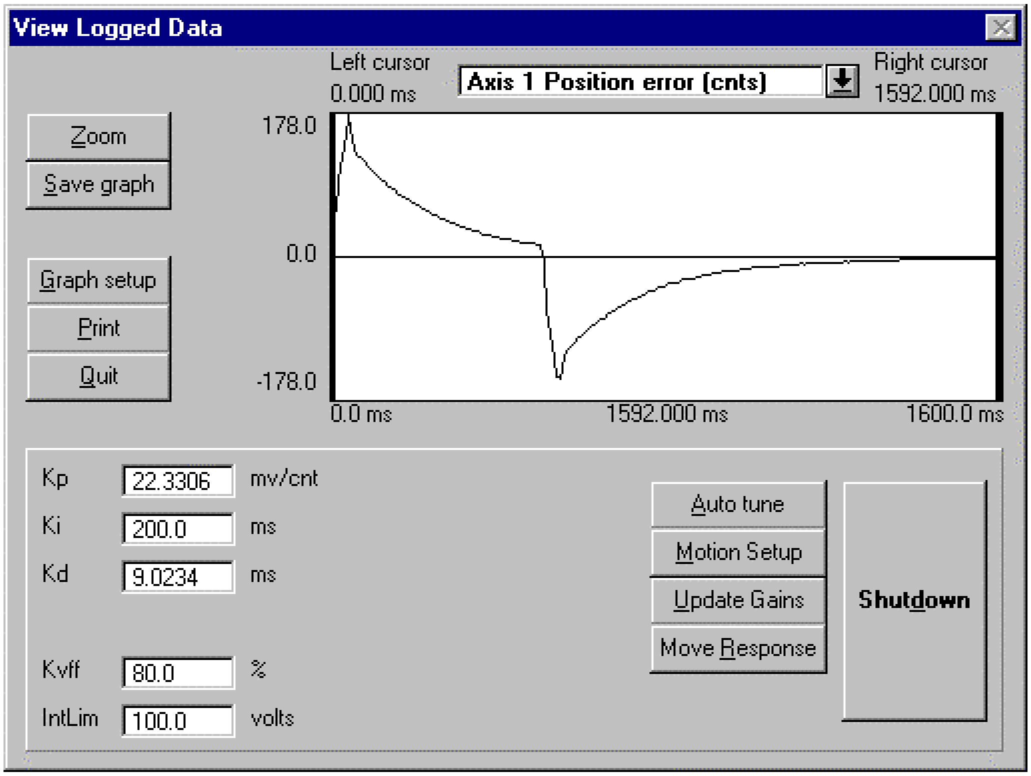
Ki 18.0467 ms

Kd 4.0 ms

Kvff 80.0 %

IntLim 100.0 volts

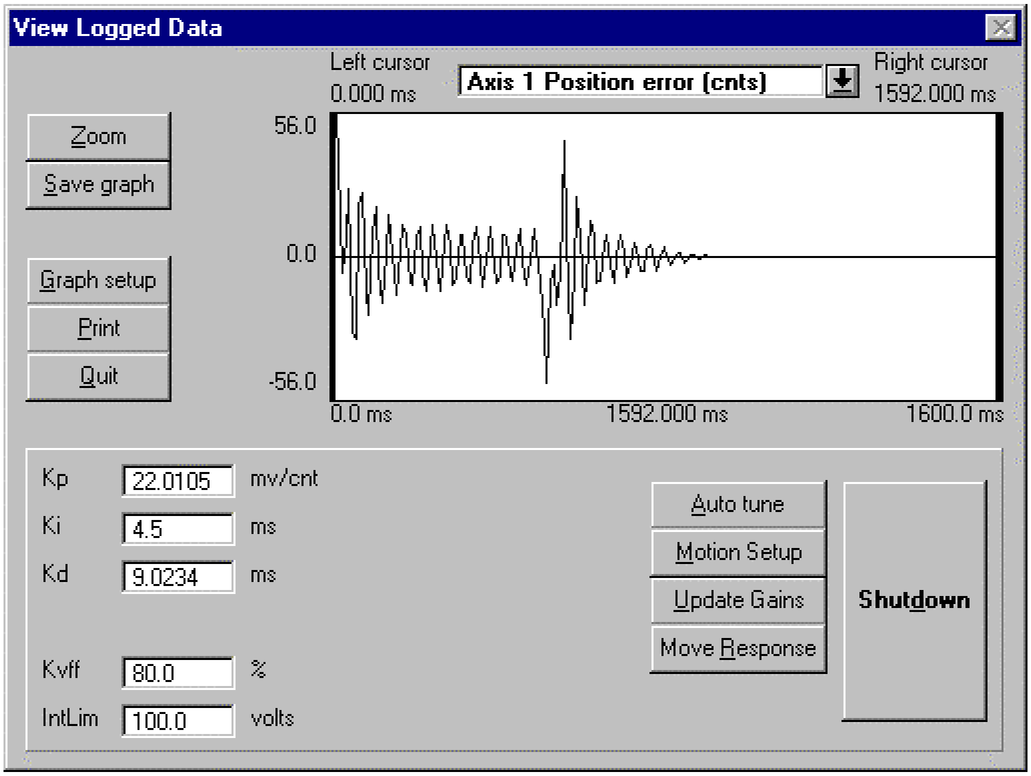
"Stiff" response with high Kp and low Kd .



Left cursor 0.000 ms
Axis 1 Position error (cnts)
Right cursor 1592.000 ms

Kp 22.3306 mv/cnt
Ki 200.0 ms
Kd 9.0234 ms
Kvff 80.0 %
IntLim 100.0 volts

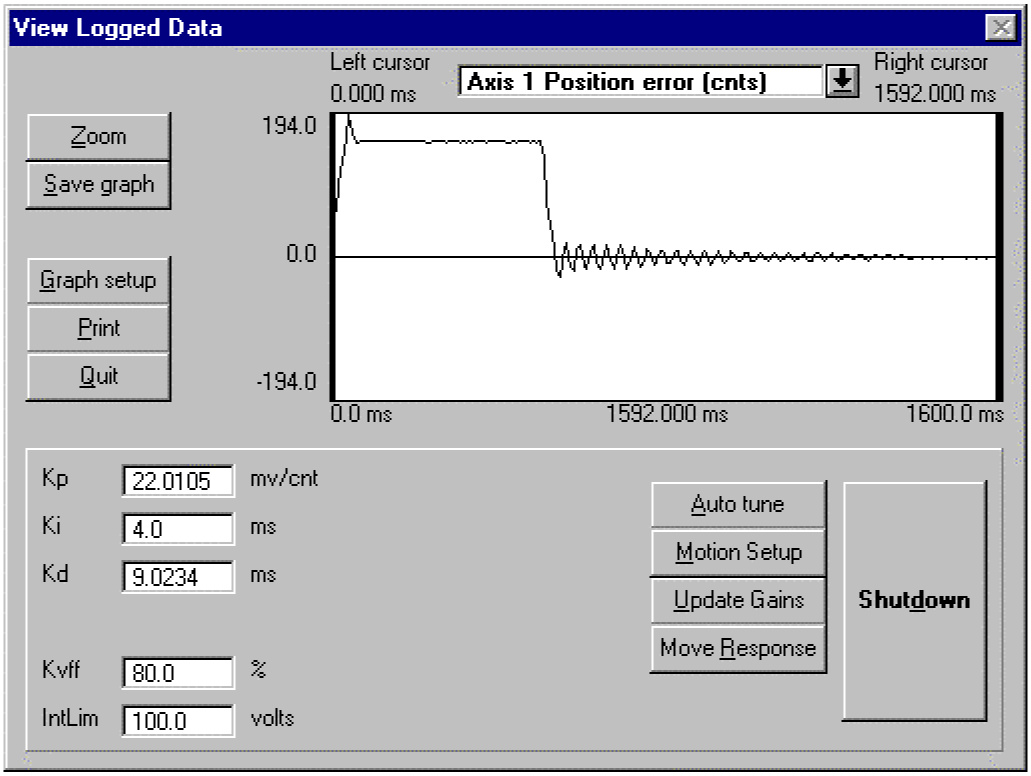
Above response with high Ki and integration enabled during motion. Note very long settling time.



Left cursor 0.000 ms
Axis 1 Position error (cnts)
Right cursor 1592.000 ms

Kp 22.0105 mv/cnt
Ki 4.5 ms
Kd 9.0234 ms
Kvff 80.0 %
IntLim 100.0 volts

Response with low Ki and integrator enabled during motion. Note excessive ringing.



View Logged Data

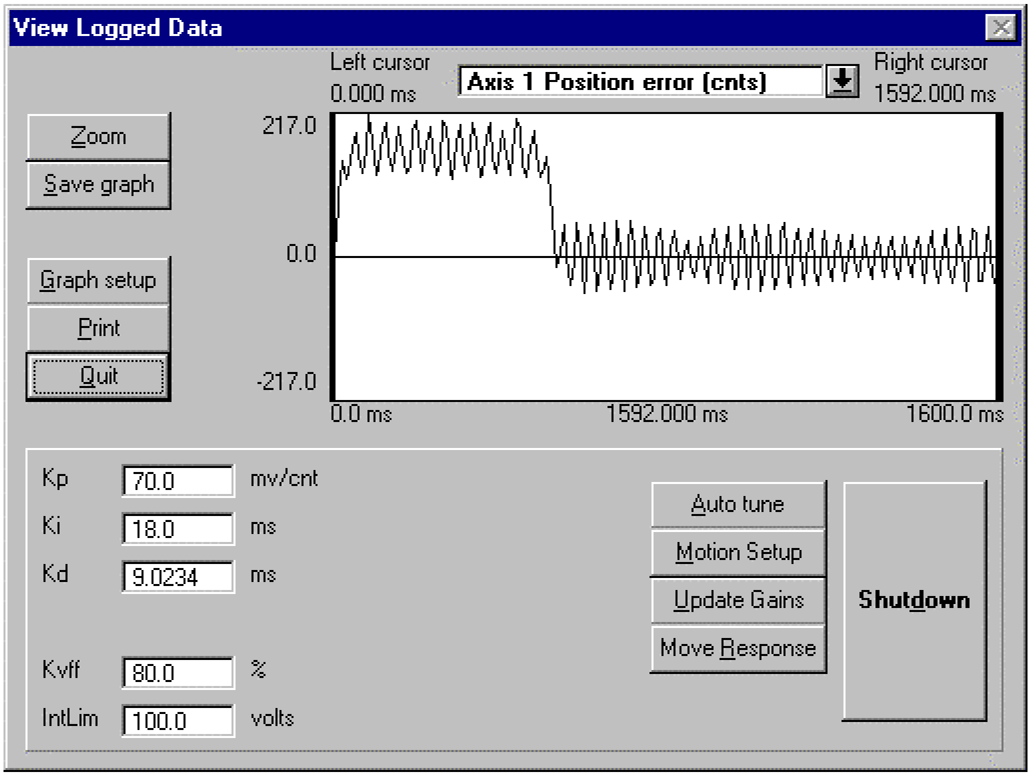
Left cursor 0.000 ms

Axis 1 Position error (cnts)

Right cursor 1592.000 ms

Kp	22.0105	mv/cnt
Ki	4.0	ms
Kd	9.0234	ms
Kvff	80.0	%
IntLim	100.0	volts

Response with low Ki and integrator disabled during motion. Note excessive ringing at end of move only. The integrator is engaged when the profile stops.



View Logged Data

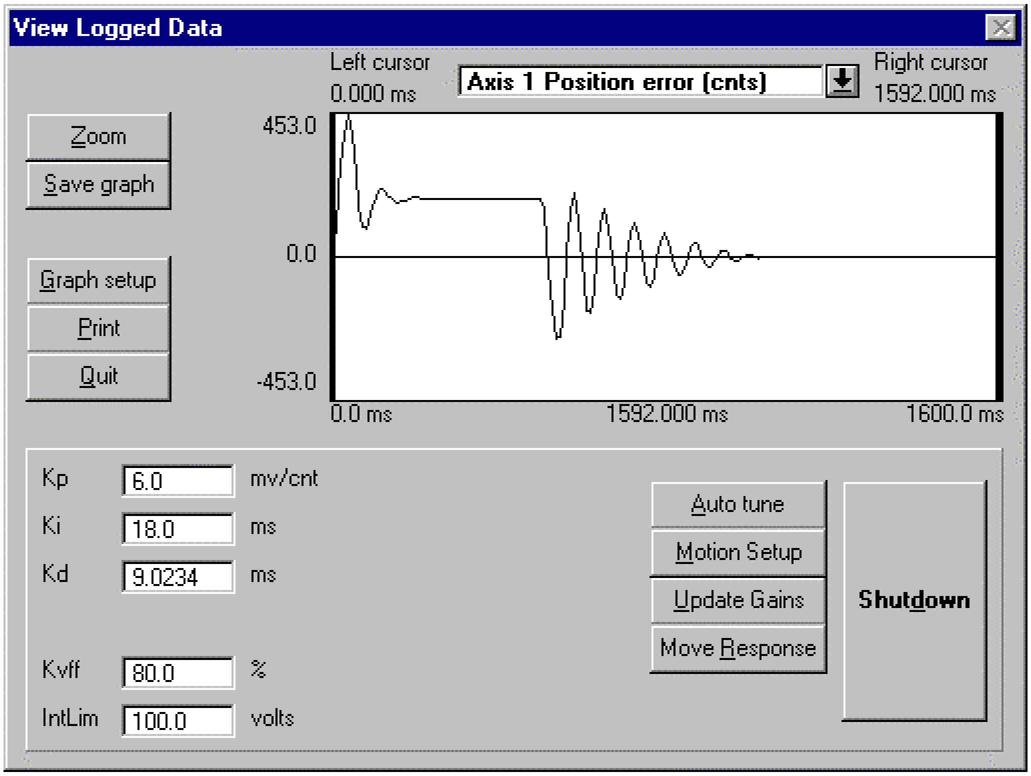
Left cursor 0.000 ms

Axis 1 Position error (cnts)

Right cursor 1592.000 ms

Kp	70.0	mv/cnt
Ki	18.0	ms
Kd	9.0234	ms
Kvff	80.0	%
IntLim	100.0	volts

Response with high Kp. Note instability can be seen as vibration. Kp should be lowered to eliminate instability.



Left cursor 0.000 ms

Axis 1 Position error (cnts)

Right cursor 1592.000 ms

Kp 6.0 mv/cnt

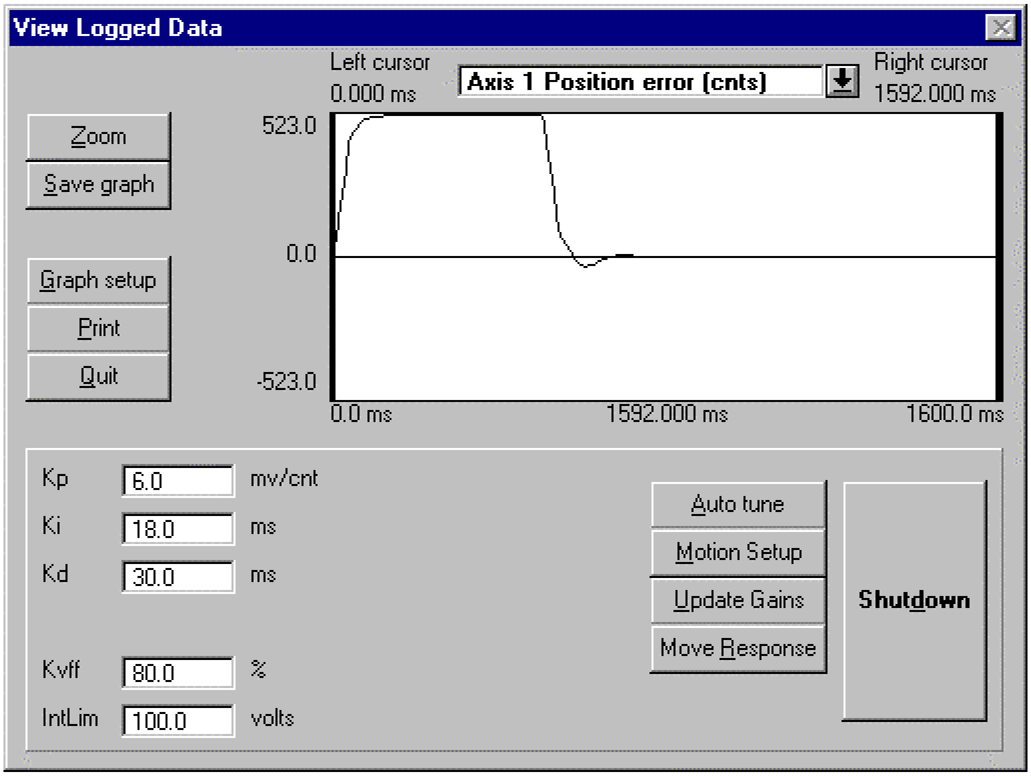
Ki 18.0 ms

Kd 9.0234 ms

Kvff 80.0 %

IntLim 100.0 volts

Response with low Kp. Note oscillation. If Kp can not be raised, Kd may be raised to reduce the ringing as shown below.



Left cursor 0.000 ms

Axis 1 Position error (cnts)

Right cursor 1592.000 ms

Kp 6.0 mv/cnt

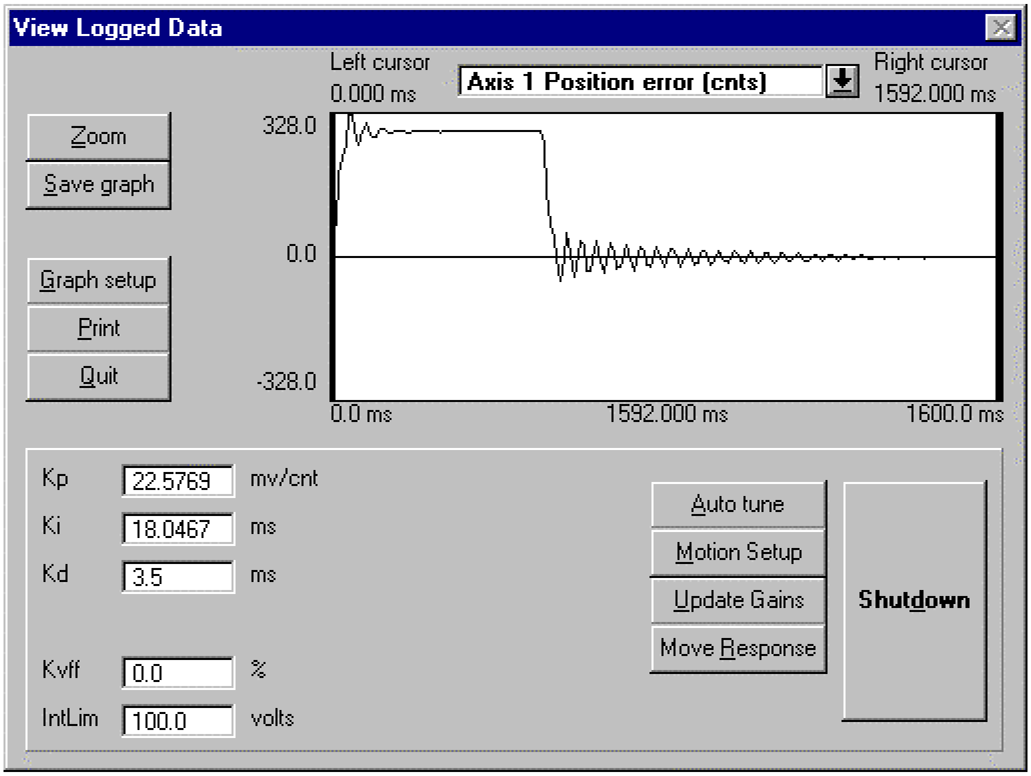
Ki 18.0 ms

Kd 30.0 ms

Kvff 80.0 %

IntLim 100.0 volts

Previous profile with Kd raised to dampen out oscillation.



View Logged Data

Left cursor 0.000 ms

Axis 1 Position error (cnts)

Right cursor 1592.000 ms

Kp 22.5769 mv/cnt

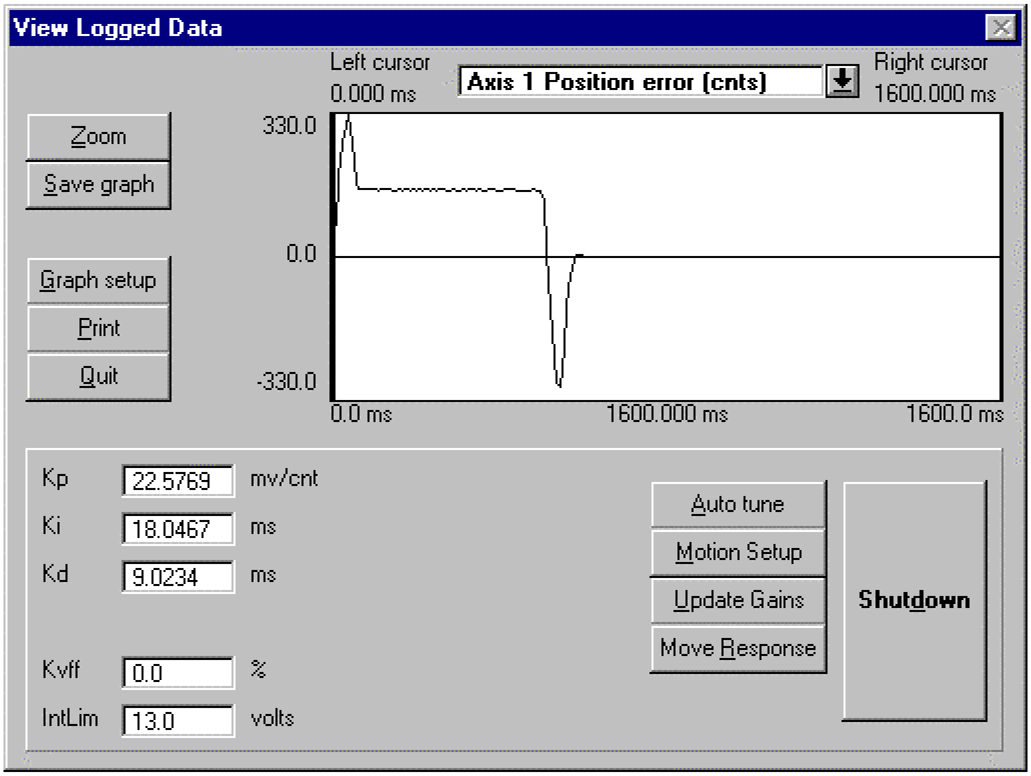
Ki 18.0467 ms

Kd 3.5 ms

Kvff 0.0 %

IntLim 100.0 volts

Response with low Kd. Note oscillation at end of profile.



View Logged Data

Left cursor 0.000 ms

Axis 1 Position error (cnts)

Right cursor 1600.000 ms

Kp 22.5769 mv/cnt

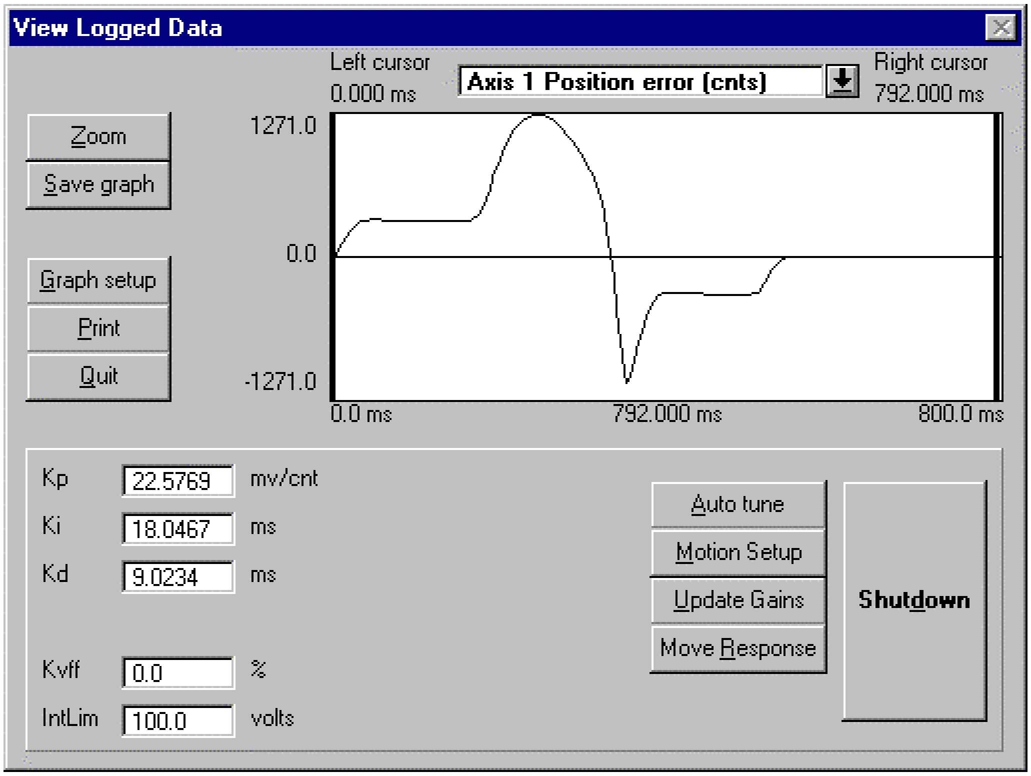
Ki 18.0467 ms

Kd 9.0234 ms

Kvff 0.0 %

IntLim 13.0 volts

Response if Intlim is too low and integrator enabled during motion. Note that the integrator cannot bring the error to zero during the flat top part of the profile.



Axis 1 Position error (cnts)

Left cursor 0.000 ms

Right cursor 792.000 ms

Kp 22.5769 mv/cnt

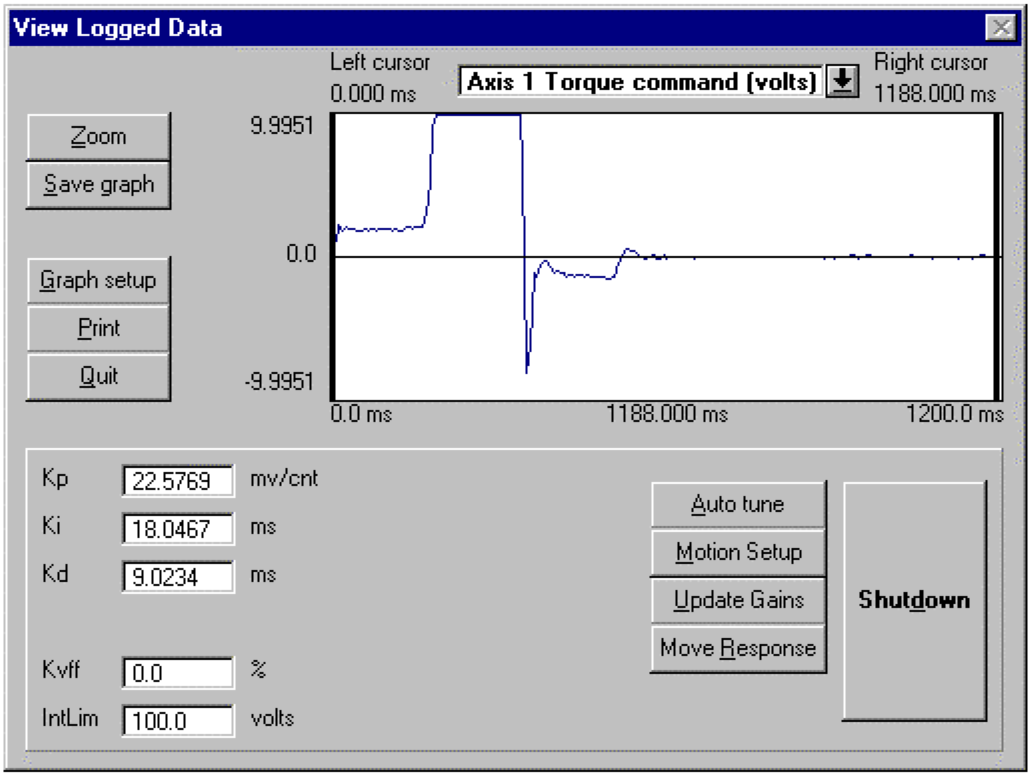
Ki 18.0467 ms

Kd 9.0234 ms

Kvff 0.0 %

IntLim 100.0 volts

Response if the programmed speed is too high for the motor. This also can be caused by the drive running at too low a bus voltage.



Axis 1 Torque command (volts)

Left cursor 0.000 ms

Right cursor 1188.000 ms

Kp 22.5769 mv/cnt

Ki 18.0467 ms

Kd 9.0234 ms

Kvff 0.0 %

IntLim 100.0 volts

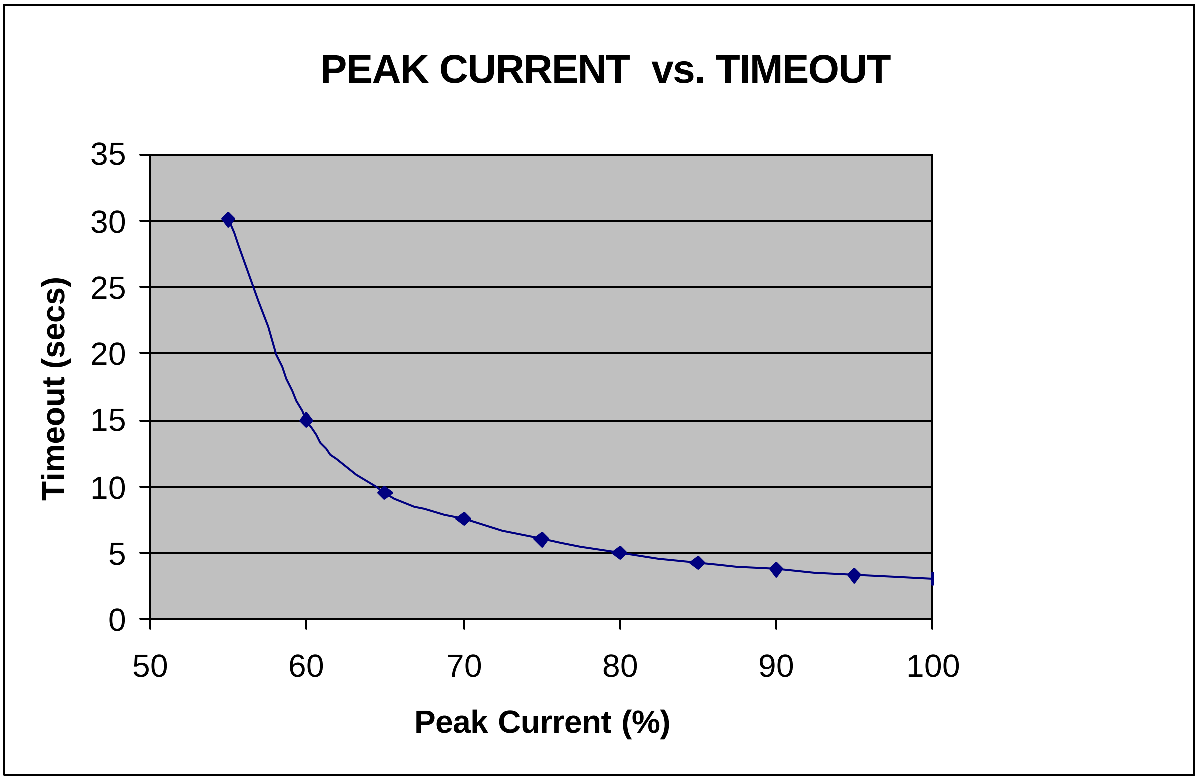
Response of the torque command for the previous profile with the speed set too high. Note that the torque command saturates at 10 volts. Any time the command goes to + or – 10V, the motor is not producing the required torque to bring the error down.

5.2.14 - Excessive Duty Cycle Shutdown

As the TDC controller responds to shaft displacement due to move commands or reaction torques, the amplifier section of the TDC produces current to drive the motor. A feature has been added that prevents the unit from generating too much current and/or motor heating due to an excessive duty cycle situation. Here, duty cycle refers to the percentage of time that the system is required to generate a current (and therefore resultant torque) above its continuous rating.

The continuous current that the drive produces is set using the current setting switches on top of the unit. The peak current available is twice the continuous setting. For example a switch setting of 4 amps continuous would result in an available peak current of 8 amps. The continuous current can be maintained indefinitely. However, currents above the continuous switch setting (up to the peak current) can only be generated for a limited length of time before damage to the TDC unit and/or the motor will result. If the unit and motor are allowed to cool (i.e. the motor rests for a short period) as a result of the current dropping below the continuous rating, then repetitive occurrences of currents above the continuous rating may be acceptable.

If an excessive duty cycle situation occurs, the amplifier will be disabled, and a fault condition will be generated. If this happens the motor shaft will spin freely unless it is held by an external brake. Error code 136 will be generated. DO NOT continuously re-enable the drive if this error persists.



Section 6.0

Software Reference Guide

6.1 Programming Commands

6.1.1 Programming Commands Grouped By Function

Motion		**Page**
BUSY	Returns the motion status of the axis.	86
CMDPOS	Returns the commanded position of the motor in units.	87
EVENT1	Sets Enable/disable and trigger state of event1.	95
EVENT2	Sets Enable/disable and trigger state of event2.	96
JOG	Run continuously in the specified direction.	111
MOTTRIG	Sets or returns the hardware trigger level for a MOVEI, MOVEA, MOVEHOME, JOG and MOVEREG cycle.	
MOVEA	Initiates an absolute indexed move.	116
MOVEHOME	Run until the home input is activated.	117
MOVEI	Initiates an incremental indexed move.	118
MOVEREG	Run until the registration input is activated, then move the specified distance.	119
STOP	Brings any motion to a controlled stop.	135
WAITDONE	Waits for motion to be done.	139
WNDGS	Enable/Disable servo drive.	140

TRAJECTORY PARAMETERS

ABSPOS	Sets or returns the absolute position.	83
ACCEL	Sets or returns the acceleration rate in units/sec/sec.	84
DECEL	Sets or returns the deceleration rate in units/sec/sec.	87
DIST	Returns the distance moved from the start of the last commanded motion or changes the move distance during indexed (MOVEA, MOVEI) motion.	89
ENCPOS	Returns the encoder absolute position.	91
ENCSPD	Returns the current speed.	92
SPEED	Sets or returns the commanded target speed.	135

SERVO PARAMETERS

FOLERR	Sets or returns the following error.	97
INTLIM	Sets or returns the servo axis integral limit.	109
KD	Sets or returns the servo axis derivative gain.	111
KI	Sets or returns the servo axis integral gain.	112
KP	Sets or returns the servo axis proportional gain.	113
KVFF	Sets or returns the velocity feed forward gain value for a servo axis.	114
OUTLIMIT	sets or returns the torque output voltage limit.	125

I/O

ANALOG	Returns the analog input voltage.	84
BCD	Returns the BCD switch value.	86
IN	Returns the discrete input state of the defined input.	106
OUT	Sets or returns the discrete output state of the defined output.	124

OVER TRAVEL LIMIT

		Page
HARDLIMOFF	Disables hard limits.	102
HARDLIMON	Enables hard limits.	103
REGLIMIT	Sets or returns the movereg limit distance.	130
SOFTLIMNEG	Sets or returns the absolute negative travel limit position.	131
SOFTLIMOFF	Disables soft limits.	132
SOFTLIMON	Enables soft limits.	133
SOFTLIMPOS	Sets or returns the absolute positive travel limit position.	134

TIME FUNCTIONS

TIMER	Sets or returns timer value.	137
WAIT	Wait for the period of time to expire.	139

PROGRAM FLOW CONTROL

DO...EXIT DO...LOOP... LOOP...UNTIL...WHILE	Begin a repeatable a block of statements.	90
END	End of program.	93
FOR...TO...EXIT FOR...NEXT	Begin a repeatable block of statements.	98
GOSUB...RETURN	Branch to a subroutine and return.	101
GOTO	Branch unconditionally to the specified label.	102
IF..THEN..ELSE..END IF	Begin a conditional block of statements.	105

INTERRUPT

INTROFFn	Disable interrupt n, where n is 1-4.	110
INTRONn	Enable interrupt n, where n is 1-4.	110
ON...INTRn	On condition go to interrupt n, where n is 1-4.	121

MISCELLANEOUS

#DEFINE	Defines a symbolic name to be a particular string of characters	88
ERR	Return error code number.	93
#INCLUDE	Includes a file name with define statements in a user task.	107
NVR	Returns or stores a REAL or INTEGER value to NVR memory.	120

BOOLEAN EXPRESSION OPERATORS

AND	Logical conjunction operator.	85
NOT	Logical complement operator.	120
OR	Logical inclusive OR operator.	123

DAISY CHAINING

UNITID	Returns the current Unit ID.	138

STRING MANIPULATION

ASC	Returns the ASCII code of character.	85
CHR$	Returns a one character string for the given ASCII code.	87
GETCHAR	Waits for a character to be received via the serial port.	100
HEX$	Returns the hex string of an integer.	104
HVAL	Returns the hex value of a string.	104

INCHAR	Returns a character from the serial port.	107
		Page
INPUT	Reads a line of data from the serial port.	108
INSTR	Returns the first occurrence of a character in a string.	108
LCASE$	Converts a string to lower case letters.	114
LEFT$	Returns the leftmost characters of a string.	115
LEN	Returns the number of characters in a string.	115
MID$	Returns the designated middle number of characters of a string.	115
PRINT	Transmit data via the serial port.	126
PRINT USING	Print string characters or formatted numbers.	127
RIGHT$	Returns the rightmost characters of a string.	130
STR$	Returns a string representation of a numeric expression.	136
STRING$	Returns a string of characters.	136
UCASE$	Converts a string to upper case letters.	137
VAL	Returns the value of a string.	138

RELATIONAL OPERATORS

=	equal to	80
<	less than	80
<= or =<	less than or equal to	80
<>	not equal to	80
>	greater than	80
=> or >=	greater than or equal to	80

ARITHMETIC OPERATORS

+	addition	80
-	subtraction or unary minus	80
*	multiplication	80
/	division 80	

ELECTRONIC GEARING

ENCPOS2	Returns the Encoder 2 position in units.	91
ENCSPD2	Returns the current Encoder 2 velocity in units/second.	92
GEAREXT	Selects the external master velocity source for gearing.	98
GEARINT	Selects the GEARVEL as the master source for gearing.	99
GEARON	Enables the master velocity source for gearing.	99
GEAROFF	Disables the master velocity source for gearing.	99
GEARRATIO	Sets or returns the external master velocity gearing ratio.	99
GEARVEL	Sets or returns the internal master velocity for gearing.	99

VARIABLE DEFINITIONS

INTEGER	var, ... , var	81
REAL	var, ... , var	81
INTEGER	var(x), ... , var(x,y)	81
REAL	var(x), ... , var(x,y)	81

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

6.1.2 Programming Commands Summary (alphabetical list)

		Page
=	equal to	80
<	less than	80
<= or =<	less than or equal to	80
<>	not equal to	80
>	greater than	80
=> or >=	greater than or equal to	80
+	addition	80
-	subtraction or unary minus	80
*	multiplication	80
/	division 80	
'	Remark	82
INTEGER var,...,var	Defines integer variable	81
REAL var, ... , var	Defines real variables.	81
A		
ABSPOS	Sets or returns the absolute position.	83
ACCEL	Sets or returns the acceleration rate in units/sec/sec.	84
ANALOG	Returns the analog input voltage.	84
AND	Logical conjunction operator.	84
ASC	Returns the ASCII code of character.	85
B		
BCD	Returns the BCD switch value.	86
BUSY	Returns the motion status of the axis.	86
C		
CHR$	Returns a one character string for the given ASCII code.	87
CMDPOS	Returns the commanded position of the motor in units.	
D		
DECEL	Sets or returns the deceleration rate in units/sec/sec.	87
#DEFINE	Defines a symbolic name to be a particular string of characters	88
DIST	Returns the distance moved from the start of the last commanded motion or changes the move distance during indexed motion.	89
DO...EXIT DO... LOOP...UNTIL... WHILE	Begin a repeatable a block of statements.	90
E		
ENCPOS	Returns the encoder absolute position.	91
ENCPOS2	Returns the Encoder 2 position in units.	91
ENCSPD	Returns the current speed.	92
ENCSPD2	Returns the current Encoder 2 velocity in units/second.	92
END	End of program.	93
ERR	Return error code number.	93
EVENT1	Sets enable/disable and trigger state of event1.	95
EVENT2	Sets enable/disable and trigger state of event2.	96

F		
FOLERR	Sets or returns the following error.	97
FOR...TO...EXIT FOR...NEXT	Begin a repeatable block of statements.	98
G		
GEAREXT	Selects the external master velocity source for gearing.	??
GEARINT	Selects the GEARVEL as the master source for gearing.	99
GEARON	Enables the master velocity source for gearing.	99
GEAROFF	Disables the master velocity source for gearing.	99
GEARRATIO	Sets or returns the external master velocity gearing ratio.	99
GEARVEL	Sets or returns the internal master velocity for gearing.	99
GETCHAR	Waits for a character to be received via the serial port.	100
GOSUB...RETURN	Branch to a subroutine and returns.	101
GOTO	Branch unconditionally to the specified label.	102
H		
HARDLIMOFF	Disables hard limits.	102
HARDLIMON	Enables hard limits.	103
HEX$	Returns the hex string of an integer.	104
HVAL	Returns the hex value of a string.	104
I		

IF..THEN..
ELSE..END IF
	Begin a conditional block of statements.	105

IN	Returns the discrete input state of the defined input.	106
INCHAR	Returns a character from the serial port.	107
#INCLUDE	Includes a file name with define statements in a user task.	107
INPUT	Reads a line of data from the serial port.	108
INSTR	Returns the first occurrence of a character in a string.	108
INTLIM	Sets or returns the servo axis integral limit.	109
INTROFFn	Disable interrupt n.	110
INTRONn	Enable interrupt n.	110
J		
JOG	Run continuously in the specified direction.	111
K		
KD	Sets or returns the servo axis derivative gain.	111
KI	Sets or returns the servo axis integral gain.	112
KP	Sets or returns the servo axis proportional gain.	113
KVFF	Sets or returns the velocity feed forward gain value for a servo axis.	114
L		
LCASE$	Converts a string to lower case letters.	114
LEFT$	Returns the leftmost characters of a string.	115
LEN	Returns the number of characters in a string.	115
M		
MID$	Returns the designated middle number of characters of a string.	115

WAIT	Wait for the period of time to expire.	139
WAITDONE	Waits for motion to be done.	139
WNDGS	Enable/disable servo drive.	140

6.1.3 SEBASIC Conventions

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

6.1.3.1 ARITHMETIC OPERATORS

The SEBASIC arithmetic operators are listed in order of precedence:

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

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

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

6.1.3.2 LOGICAL OPERATORS

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

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

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

6.1.3.3 RELATIONAL OPERATORS

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

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

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

6.1.3.4 BASIC DATA TYPES

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

The following are examples of some REAL values:

```
        +1.524          -100.1          2.1e-4
```

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

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

The following are examples of some INTEGER values:

```
        +1              -100            -3487
```

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

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

STRING values can be any ASCII character. A list of ASCII characters is provided in Section 9, Glossary.

RULES FOR INTEGER DIVISION:

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

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

begin:                                          › begin program

        num1    = 10                            › set INTEGER num1 equal to 10
        denom1 = 4                              › set INTEGER denom1 equal to 4
        denom2 = 4                              ›set REAL denom2 equal to 4

        answer1  = num1/denom1                  › use division operator to divide num1 by denom1
        answer2= num1/denom2                    ›use division on num1 by denom2

end                                             ›end program
```

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

```
REAL x          ›declare x as a REAL variable

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

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

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

6.1.3.5 CASE SENSITIVITY IN STATEMENTS & COMMANDS

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

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

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

6.1.3.6 CALCULATIONS USING TRAJECTORY PARAMETERS AND VARIABLES

Caution must be used when performing calculations based on the Trajectory Parameters, **ABSPOS, ACCEL, DECEL, DIST, ENCPOS, ENCSPD**, and **SPEED**. Comparisons of values returned directly from reading these parameters or values of variables calculated from these parameters may not always yield the expected results. The reason for this is that digital systems have inherent resolution limitations. In the case of digital servos, the actual position of the motor shaft at any time can only be represented within one encoder count. If the user is programming in units, the actual position is calculated based on the number of encoder count per user unit. Following example illustrates the need to use caution when using these parameters.

Let us say that the system has an encoder resolution of **4000 counts per motor revolution** (typical of a 1000 line encoder with 4X phase quadrature decoding). Let us also assert that the user wishes to program his system in **revolutions**. In this case, the *Configuration and Setup* parameter units per motor revolution would be set to **1**, and the encoder line count would be set to **1000 lines/rev**.

Let=s assume that the user wishes to perform a move of **1.5238 revolutions**, the system will move the servo to the nearest encoder count to this position. In this case,

(4000 counts/rev) x (1.5238 revs) = 6095.2 counts

and since the system can only resolve to 1 count, the fractional portion is dropped, yielding an actual position of **6095 counts**. If the **ENCPOS** command is now used to read back the motor=s actual position, the result will be

(6095 counts)/ (4000 counts/rev) = 1.52375.

It is important to note that the programmed move, **1.5238 is NOT exactly the same as 1.52375.** Also, any variable based on the **ENCPOS**, for example if the program sets a REAL variable

x = 15 * ENCPOS

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

6.1.3.7 PROGRAM COMMENTS

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

EXAMPLES:	'MOVEI=10	›The program will not execute this line
	MOVEI=100	›The program will execute this line

6.1.4 Programming Commands - Alphabetical Listing

ABSPOS

Trajectory Parameters

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

PROGRAM SYNTAX:

ABSPOS=expression
ABSPOS - used in an expression

REMARKS:

ABSPOS=expression
Sets the absolute position in units.

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

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

EXAMPLES:

ABSPOS=2
sets absolute position to 2 units.

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

Trajectory Parameters

ACCEL

ACTION:

Sets or returns the acceleration value of the motor.

PROGRAM SYNTAX:

ACCEL=expression
ACCEL - used in an expression

REMARKS:

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

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

The rate at which the motor speed is increased. Specifying a 0 or negative value will result in error code 6. Specifying a value greater than "Max Accel" set in the system *Configuration and Setup* will result in ACCEL being set to "Max Accel". At power up ACCEL is initialized to 50% of "Max Accel". ACCEL can be set during motion, but the new setting will not be used until the next move. Reading ACCEL returns the most recent setting.

EXAMPLES:

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

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

I/O Functions

ANALOG

ACTION:

Returns the analog input value in volts.

PROGRAM SYNTAX:

ANALOG - used in an expression

REMARKS

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

EXAMPLES:

variable x to the analog input voltage.

AND

Boolean Operator

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

PROGRAM SYNTAX: expression1 AND expression2

REMARKS: The AND operator uses this "truth table":

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

The result is true if both expressions are true.

EXAMPLES:

if (x >2 AND y < 3) then goto INDEX

›The controller checks to see if x > 2 **and** y < 3. If both conditions are true the program goes to a label called INDEX.

ASC

String Manipulation

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

PROGRAM SYNTAX: ASC(n$)

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

EXAMPLES:

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

BCD

I/O Operator

ACTION: Returns the value on the BCD switches.

PROGRAM SYNTAX: BCD - used in an expression

REMARKS: BCD

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

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

EXAMPLES:

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

BUSY

Motion

ACTION: Returns the motion status.

PROGRAM SYNTAX: BUSY - used in an expression

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

EXAMPLES:

```
                        ›prints system absolute position
        PRINT#1,ABSPOS  ›while motion is still occurring
LOOP
```

CHR$

String Manipulation

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

PROGRAM SYNTAX: CHR$(code)

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

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

CMDPOS

Motion

ACTION: Returns the commanded position of the motor in units.

PROGRAM SYNTAX: CMDPOS – used in an expression

REMARKS: This command is defined in more details in the Gearing section of the manual.

DECEL

Trajectory Parameters

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

PROGRAM SYNTAX:

DECEL=expression
DECEL - used in an expression

REMARKS:

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

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

The rate at which the motor speed is decreased. Specifying a 0 or negative value will result in error code 7. Specifying a value greater than "Max Accel" set in the *Configuration and Setup* will result in DECEL being set to "Max Accel". At power up DECEL is initialized to 50% of "Max Accel". DECEL can be set during motion, but the new setting will not be used until the next move. Reading DECEL returns the most recent setting.

EXAMPLES: 3.1 units/sec^2.

X = DECEL

Sets variable X equal to the value of deceleration.

#DEFINE

Miscellaneous Command

ACTION:

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

PROGRAM SYNTAX:

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

REMARKS:

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

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

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

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

EXAMPLES:

```
#DEFINE TRUE 1
```

Substitutes a 1 when the name TRUE is encountered.

```
#DEFINE FALSE 0
```

Substitutes a 0 when the name FALSE is encountered.

```
#DEFINE SENDPOS @1 PRINT#@1,ABSPOS
```

Sends the absolute position via port @1.

```
SENDPOS 1
```

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

```
SENDPOS 2
```

Sends the absolute position via port #2. The 2 is substituted for the @1.

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

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

DIST

ACTION:

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

PROGRAM SYNTAX:

DIST = expression
DIST - used in an expression

REMARKS:

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

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

EXAMPLES:

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

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

DO...LOOP

ACTION:

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

PROGRAM SYNTAX 1:

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

PROGRAM SYNTAX 2:

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

PROGRAM SYNTAX 3:

DO <spc> : <spc> LOOP ....

REMARKS:

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

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

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

EXAMPLES:

DO WHILE EVENT1 <> 1 ›Continue the loop **while** 'event1 does not equal 1.
:
LOOP ›End of loop.

ENCPOS

Trajectory Parameters

ACTION:

Returns the encoder position.

PROGRAM SYNTAX:

ENCPOS - used in an expression

REMARKS:

ENCPOS
Evaluates and returns the present encoder position.

The actual motor position. 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

EXAMPLES:

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

ENCPOS2

Trajectory Parameters

ACTION:

Returns the Encoder 2 position in user units.

PROGRAM SYNTAX:

ENCPOS2 – used in an expression

REMARKS:

This command is defined in more details in the Gearing section of the manual.

ENCSPD

Trajectory Parameters

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

PROGRAM SYNTAX: ENCSPD - used in an expression

REMARKS: ENCSPD
Evaluates and returns the current encoder speed.

Reading ENCSPD returns the actual motor speed with a resolution of:

(250 * "Units/rev")/("Line count" * "Servo Sample Time") units/sec.

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

Example: "Units/rev" = 1 , "Line count" = 1000, "Servo Sample Time" = 1

example resolution = .25 rev/sec

Note: Limit = 2 meg counts/sec. at 1ms sample time
= 1 meg counts/sec. at 2 ms sample time

The returned motor speed value is a signed number.

EXAMPLES:

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

ENCSPD2

Trajectory Parameters

ACTION: Returns the current Encoder 2 velocity in units/second.

PROGRAM SYNTAX: ENCSPD2 – used in an expression

REMARKS: This command is defined in more detail in the Gearing section of the manual.

END

Program Flow Control

ACTION: Signifies the end of a program.

PROGRAM SYNTAX: END

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

EXAMPLES:

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

ERR

Return Error Code

ACTION: Returns the error status of the controller.

PROGRAM SYNTAX: ERR - used in an expression

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

Error Code	Description
0	No error.
1	Could not burn flash successfully.
2	Could not download file.
3	Not enough memory to execute user program.
4	Attempt to access a non-existent array element.
5	Real data too large to convert to Integer data.
6	Attempt to set accel data <= 0.
7	Attempt to set decel data <= 0.
8	Attempt to access non-existent output.
9	Attempt to access non-existent input.
10	Attempt to divide by 0.
11	Received serial data will not fit in buffer.
12	Motion occurring when program ended.
13	Attempt to execute user program that is not present.
14	Incorrect user program checksum.

15	Attempt to set Kp out of range.
16	Attempt to set Kd out of range.
17	Attempt to set Ki out of range.
18	Attempt to set Kvff out of range.
19	Attempt to set FOLERR <0.
20	Attempt to set INTLIM out of range.
21	Move distance too large.
22	Function not implemented.
23	INPUT command error occurred
24	NVR specified location out of range (1-400).
25	NVR device not detected.
128	+limit switch activated.
129	-limit switch activated.
130	+ Software travel limit exceeded.
131	- Software travel limit exceeded.
132	This code is reserved for future use.
133	Excessive position error.
134	Registration distance too small.
135	Attempt to move with drive not enabled.
136	Excessive duty cycle shutdown or attempt to move with drive not ready.
137	Gearing backlog overflow.
138	NVR data corrupt

If the hard limit inputs are enabled during gearing motion and the hard limit input becomes active an error will occur. The error code for the + Limit is 128 and 129 for the – Limit.

If the Soft limits are enabled during gearing motion and the soft limit is exceeded an error will occur. The error code for the + Software travel limit is 130 and 131 for the – Software travel limit.

When operating in electronic gearing mode, a hard or soft limit error will force a GEAROFF condition before jumping to the error handler routine.

Gearing motion can backlog counts when the input rate exceeds the imposed rate limit for gearing. No error will occur if this count is less than 32767 counts. However, if this count is exceeded an error will occur which will cause a GEAROFF condition before going to the error handler routine. The error code for a backlog overflow is 137.

If there is no error handler routine present in the program, an error will simply terminate program execution, otherwise an error causes the program to jump to the error handler routine (label **ERROR_HANDLER**). The error handler routine can not be interrupted. The error handler routine is terminated with either an END statement or a GOTO <label> statement. The END statement will terminate program execution. The GOTO <label> statement will cause program execution to continue. At this point the program can be interrupted.

EXAMPLES:

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

EVENT1

ACTION:

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

PROGRAM SYNTAX:

EVENT1=expression

REMARKS:

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

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

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

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

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

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

EXAMPLES:

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

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

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

EVENT2

ACTION:

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

PROGRAM SYNTAX:

EVENT2=expression

REMARKS:

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

The EVENT2 triggering for a MOVEHOME or MOVEREG cycle is as-signed in the user program Configuration and Setup.

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

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

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

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

EXAMPLES:

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

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

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

FOLERR

ACTION:

Sets or returns the position error limit. When the position error limit is exceeded, the windings are turned off, the analog output voltage is set to zero and an error is set.

PROGRAM SYNTAX:

FOLERR=expression
FOLERR - used in an expression

REMARKS:

FOLERR= expression
Sets the position error limit in units. Setting the position error limit to zero sets the position error limit to infinity.

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

FOLERR sets or reads the "Following Error" Limit. "Following Error" is set initially in the *Configuration and Setup* and is the absolute value of the difference between the commanded and actual motor position, i.e. |ABSPOS - ENCPOS|. The test for excessive "Following Error" is only performed if the drive enable output is on and the FOLERR setting is not zero. If the "Following Error" exceeds the FOLERR setting, then the error code is set to 133, the windings are turned off and the analog output voltage is set to zero. FOLERR is limited to the number of user units corresponding to 32767 encoder counts. FOLERR is initialized to the number of units corresponding to 32767 encoder counts at power up. A negative setting for FOLERR results in error code 19. If an attempt is made to set FOLERR greater than 32767, the FOLERR is set to its maximum value of 32767. Reading FOLERR returns the present setting in user units.

EXAMPLES:

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

FOR...NEXT

Program Flow Control

ACTION:

Repeats a block of statements a specified number of times.

PROGRAM SYNTAX:

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

REMARKS:

Counter is a variable used as the loop counter.
Start# is the initial value of the counter.
End# is the ending value of the counter.
The step size is always 1.

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

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

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

EXAMPLES:

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

GEAREXT

Gearing

ACTION:

This command selects the External master velocity source for gearing which is the Encoder 2 input port.

PROGRAM SYNTAX:

GEAREXT

REMARKS:

This command is defined in more details in the Gearing section of the manual.

GEARINT

Gearing

ACTION: Selects the internal master velocity source, GEARVEL command, for gearing.

PROGRAM SYNTAX: GEARINT

REMARKS: This command is defined in more detail in Section 8, Gearing .

GEARON

Gearing

ACTION: Enables the master velocity for the Gearing mode of Operation.

PROGRAM SYNTAX: GEARON

REMARKS: This command is defined in more detail in Section 8, Gearing.

GEAROFF

Gearing

ACTION: Disables the master velocity for the Gearing mode of Operation.

PROGRAM SYNTAX: GEAROFF

REMARKS: This command is defined in more detail in Section 8, Gearing.

GEAR RATIO

Gearing

ACTION: Selects the external follower motor to master encoder gearing ratio. The ratio is follower motor revs / master encoder revs.

PROGRAM SYNTAX:

GEARRATIO =
GEARRATIO – used in an expression

REMARKS: This command is defined in more detail in Section 8, Gearing.

GEARVEL

Gearing

ACTION: Sets or returns the master velocity for internal gearing in units/second.

PROGRAM SYNTAX:

GEARVEL = expression
GEARVEL – used in an expression

REMARKS: This command is defined in more detail in Section 8, Gearing.

GETCHAR

ACTION:

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

PROGRAM SYNTAX:

GETCHAR(n) - used in an expression

REMARKS:

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

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

EXAMPLES:

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

GOSUB... RETURN

ACTION:

Branches to, and returns from, a subroutine.

PROGRAM SYNTAX:

GOSUB [linelabel]

REMARKS:

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

Subroutines can only be nested ten deep.

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

Subroutines can appear anywhere in the program (except within an interrupt routine). It is good programming practice to make them readily distinguishable from the main program.

EXAMPLES:

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

GOTO

Program Flow Control

ACTION:

Branches unconditionally to the specified label.

PROGRAM SYNTAX:

```
 GOTO [label]
```

REMARKS:

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

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

EXAMPLES:

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

HARDLIM OFF

Over Travel Limit

ACTION:

Disables the hardware limit inputs.

PROGRAM SYNTAX:

```
HARDLIMOFF
```

REMARKS:

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

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

EXAMPLES:

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

HARDLIM ON

Over Travel Limit

ACTION:

Enables the hardware limit inputs.

PROGRAM SYNTAX:

HARDLIMON

REMARKS:

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

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

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

The state of the +Limit can be read with the IN(3) command and the state of the -Limit can be read with the IN(4) command.
The relation between the return value from the IN command and the limit status is shown in the following table.

active signal level	IN command	limit status
switch closing	0	not active
	1	active
switch opening	0	active
	1	not active

EXAMPLES:

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

HEX$

String Manipulation

ACTION: Returns the hex string of an Integer value.

PROGRAM SYNTAX: A$=HEX$(expression)

REMARKS: The expression must be an integer value.

EXAMPLES:

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

HVAL

String Manipulation

ACTION: Returns the decimal value of a hexadecimal string.

PROGRAM SYNTAX: x=HVAL(A$)

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

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

EXAMPLES:

```
x=HVAL("0XFF")                  ' x is set to 255

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

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

ACTION:

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

PROGRAM SYNTAX 1:

IF condition THEN thenpart [ELSE elsepart]

PROGRAM SYNTAX 2:

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

REMARKS:

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

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

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

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

EXAMPLES:

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

IN

I/O Operator

ACTION:

Returns the state of a digital input.

PROGRAM SYNTAX:

IN(nn) - used in an expression

REMARKS:

nn is the specified digital input 1-19.

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

The inputs are assigned as follows:

Input	Signal Designation	Input	Signal Designation
1	Event1/IN1	11	BCD0/IN11
2	Event2/IN2	12	BCD1/IN12
3	"+Limit"/IN3	13	BCD2/IN13
4	"-Limit"/IN4	14	BCD3/IN14
5	"Run"/IN5	15	BCD4/IN15
6	"Clear"/IN6	16	BCD5/IN16
7	AFeedhold@/IN7	17	BCD6/IN17
8	IN8	18	BCD7/IN18
9	IN9	19	DRV Ready
10	IN10		

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

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

EXAMPLES:

IF IN(6)=1 then goto continue

INCHAR

String Manipulation

ACTION:

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

PROGRAM SYNTAX:

INCHAR(n)

REMARKS:

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

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

EXAMPLES:

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

#INCLUDE

Miscellaneous Command

ACTION:

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

PROGRAM SYNTAX:

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

REMARKS:

Drive is the root directory of the drive.

Subdir is the path required to find the file.

Filename is the include filename with extension .inc.

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

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

EXAMPLES:

```
#INCLUDE  c:\mcpi\iws.inc  ' include file iws.inc
```

INPUT

String Manipulation

ACTION:

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

PROGRAM SYNTAX:

```
INPUT#1,n$
INPUT#1,n$,var1$[,var2$] ... [var_n$]
INPUT#1,x
INPUT#1,x,y[,z] ... [,z1]
INPUT#2,n$
INPUT#2,n$,var1$[,var2$] ... [var_n$]
INPUT#2,x
                              INPUT#2,x,y[,z] ... [,z1]
```

REMARKS:

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

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

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

EXAMPLES:

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

Program:

```
string          a$
integer  x
real            y
INPUT#2,a$,x,y               ' sets a$="A555555"
                             'sets x=100
                             'sets y=10.5
```

INSTR

String Manipulation

ACTION:

Returns the character position of the first occurrence of a specified string in another string.

PROGRAM SYNTAX:

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

REMARKS:

The expression must be an integer variable.

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

EXAMPLES:

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

INTLIM

the value of 4 is returned.

ACTION:

Sets the Integral limit for the controller. This is the limit of the contribution to the servo output from the integral of the position error.

PROGRAM SYNTAX:

INTLIM=expression
INTLIM - used in an expression

REMARKS:

The expression is the Integral Limit of the servo axis in volts.

Limits the contribution of the integral term to the servo loop's output. INTLIM can be set between 0 and 319.99 volts. Setting it to a value outside this range will result in error number 20. If the input value is out of range, the previous setting is retained. Reading INTLIM returns the present setting in volts.

INTLIM=expression
Sets the Integral limit to the expression value.

INTLIM
Returns the integral limit value of an axis.

EXAMPLES:

```
INTLIM=5      'Sets the integral limit to 5 volts.

x=INTLIM      'Sets variable x to the integral limit.
```

INTROFFn

Interrupt

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

PROGRAM SYNTAX: INTROFFn

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

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

EXAMPLES:

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

INTRONn

Interrupt

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

PROGRAM SYNTAX: INTRONn

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

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

EXAMPLES:

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

JOG

Motion

ACTION: Jog the motor in a specified direction.

PROGRAM SYNTAX: JOG = expression

REMARKS: JOG=expression

The sign of the expression determines the direction of motion. If the expression is positive or 0, jogging will take place in the positive direction.

If the expression is negative, jogging will take place in the negative direction. The speed of the jog move is determined by the last SPEED command.

Use the STOP command for stopping the motor.

KD

Servo Parameters

ACTION: Sets or returns the derivative gain for the servo motor.

PROGRAM SYNTAX: KD=expression

REMARKS: KD - used in an expression

The expression is the derivative gain value of the servo axis.

The **Units** are milliseconds.

The expression value must be positive.

KD must be non-zero for system stability. Setting KD affects the gains for the velocity and feedforward terms. A negative value will result in error code 16 as would a value that causes any of the affected gains to exceed their upper bounds. When a setting for KD results in error code 16, the previous gain settings are retained. Reading KD returns the present setting.

EXAMPLES:

```
KD=4          'Sets the derivative gain to 4 milliseconds.
x=KD          'Sets variable x to the derivative gain.
```

KI

ACTION:

Sets or returns the Integral gain of the servo motor.

PROGRAM SYNTAX:

KI=expression
KI - used in an expression

REMARKS:

The expression is the integral gain value of the servo axis.
The **Units** are milliseconds.

The expression value must be positive.

KI, which is specified in ms., determines how fast the integral term grows with non-zero position error. The growth rate is inversely related to the value of KI. For example the integral term grows 5 times faster with KI = 10 than with KI = 50. A special case is KI = 0, which disables the integral action and sets the integral term to zero. When the drive is disabled, the integral term is set to zero. Setting KI only affects the gain for the integral term. A negative value will result in error code 17 as would a small enough positive value. Most likely, the system would go unstable well before a KI setting results in a range error. When an out of range error occurs, the previous gain setting is retained. Reading KI returns the present setting.

EXAMPLES:

```
KI=1            'Sets the integral gain to 1 millisecond.
x=KI            'Sets variable x to the integral gain.
```

KP

ACTION:

Sets or returns the proportional gain of the servo motor.

PROGRAM SYNTAX:

KP=expression
KP - used in an expression

REMARKS:

The expression is the proportional gain value of the servo axis.
The **Units** are millivolts/encoder count.

The expression value must be positive.

KP determines the size of the proportional term for a given position error. The units of KP are millivolts per encoder count. A negative value will result in error code 15 as would a value that causes any of affected gains to exceed their upper bounds. When a setting for KP results in error code 15, the previous gain settings are retained. Reading KP returns the present setting.

EXAMPLES:

```
KP=50 'Sets the proportional gain to 50 millivolts/encoder count (.05 volts/encoder count).
x=KP            'Sets variable x to the proportional gain value.
```

KVFF

Servo Parameters

ACTION:

Sets or returns the velocity feed forward gain value for the servo motor.

PROGRAM SYNTAX:

KVFF=expression
KVFF - used in an expression

REMARKS:

KVFF, which is specified in % can be used to reduce the position error during motion. It does not affect system stability. The minimum error occurs with KVFF near 100%. Setting KVFF only affects the gain for the feedforward term. Setting KVFF less than 0 will result in error code 18. Setting KVFF above 200 % will result in the feedforward gain exceeding its upper bound and error E_KVFF_RANGE being set. When setting KVFF and error code 18 results, the previous gain setting is retained. Reading KVFF returns the present setting.

The expression value must be positive.

EXAMPLES:

```
KVFF=100     'Sets the velocity feed forward gain  to 100%.
x=KVFF       'Sets variable x to the velocity feed forward gain value.
```

LCASE$

String Manipulation

ACTION:

Converts and returns a string with lower case letters.

PROGRAM SYNTAX:

string1$=LCASE$(string2$)

REMARKS:

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

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

EXAMPLES:

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

LEFT$

String Manipulation

ACTION: Returns the leftmost characters of a string.

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

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

EXAMPLES:

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

LEN

String Manipulation

ACTION:

ters in the designated string.

PROGRAM SYNTAX:

xpression

REMARKS:

integer type. If the input string is a null string returns a 0.

EXAMPLES:

```
 ' sets A=4
```

MID$

String Manipulation

le number of characters of a string.

ACTION:

t,number)

PROGRAM SYNTAX:

g position of the input string (string2$).

REMARKS:

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

EXAMPLES:

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

MOTTRIG

Motion

ACTION:

Sets or returns the hardware trigger level for a MOVEI, MOVEA, MOVEHOME, JOG or MOVEREG cycle.

PROGRAM SYNTAX:

MOTTRIG = expression
MOTTRIG – used in an expression

REMARKS:

Allows hardware inputs 1 or 2 to trigger motion. This results in a quick response time from the input trigger to the start of motion. The maximum start delay using the MOTTRIG command is 1 millisecond.

The expression specifies the starting trigger for the above motion commands and is specified as follows:

0 no trigger required (default)
1 start motion on event 1 active
2 start motion on event 2 active
3 start motion on event 1 inactive
4 start motion on event 2 inactive

If the expression is not a value 0-4 MOTTRIG will become 0.

The default is no trigger required when program execution starts.

This command is ignored if commanded motion is taking place. This command is defined in more detail in Section 8, Gearing

EXAMPLES:

```
MOTTRIG = 1           ‘ event 1 active trigger required to start motion
MOVEI = 1             ‘ move 1 unit after event 1 goes active
DO : LOOP UNTIL BUSY = 1 ‘ wait for motion start
 WAITDONE

 INTEGER trigger
 trigger = MOTTRIG    ‘ sets variable trigger equal to the current MOTTRIG
                        value.
```

MOVEA

Motion

ACTION:

Initiates the motor to move to the specified absolute position.

PROGRAM SYNTAX:

MOVEA=expression

REMARKS:

The expression represents the specified absolute position.

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

EXAMPLES:

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

MOVE HOME

ACTION:

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

PROGRAM SYNTAX:

MOVEHOME=expression

REMARKS:

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

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

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

EXAMPLES:

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

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

MOVEI

ACTION: Initiate an incremental move.

PROGRAM SYNTAX: MOVEI=expression

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

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

EXAMPLES:

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

MOVEREG

ACTION:

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

PROGRAM SYNTAX:

MOVEREG=expression

REMARKS:

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

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

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

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

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

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

EXAMPLES:

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

NOT

Boolean Operator

ACTION: The logical NOT operator is used in boolean expressions.

PROGRAM SYNTAX: NOT expression

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

expression	condition result
True	False
False	True

EXAMPLES:

```
Do
        :
        :
Loop while (NOT (x=1))
```

›The controller will continue to execute until variable X **does not** equal 1.

NVR

NVR Memory

ACTION: Returns or stores a REAL or INTEGER value to NVR memory.

PROGRAM SYNTAX:

```
NVR(element) – used in an expression
NVR(exp1)      – used in an expression
NVR(element) = expression
NVR(exp1)      = expression
```

REMARKS:

element or **exp1** defines the NVR location being addressed (1-400).

expression is the value being stored in the specified NVR location.

The NVR has 400 location for storage.

To clear all 400 locations of NVR use host command **CLRNVR**.

EXAMPLES:

```
STRING data$
PRINT#2,"load Speed value"
INPUT#2,data$                          ' input speed value
NVR(1)=VAL(data$)                      ' save speed value
PRINT#2,"load move distance value"
INPUT#2,data$                          ' input move value
NVR(2)=VAL(data$)                      ' save move value
SPEED=NVR(1)
MOVEI=NVR(2)
```

```
DO WHILE BUSY : LOOP          ‘ wait for motion done
```

ON...INTRn

ACTION: Sets condition to execute subroutine INTRn.

PROGRAM SYNTAX: ON [condition] INTRn

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

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

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

Note: Other subroutines **CANNOT** be called from within the interrupt routine.

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

ON <condition 1> **INTR1**
ON <condition 2> **INTR2**

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

NEXT PROGRAM STATEMENT	'execute next line in program
program checks <condition 1>	'check condition 1, if it's TRUE jump to code at INTR1: If not, continue and
program checks <condition 2>	'check condition 2, if it's TRUE jump to INTR2: If not execute next program line.
INTROFF1	
program checks <condition 2>	'check condition 2, if it's TRUE jump to INTR2: If not execute next program line.
INTROFF2	'no conditions are checked since all interrupts have been disabled.
NEXT PROGRAM STATEMENT	'execute next line in program no conditions are checked since both interrupt 1 and interrupt 2 were disabled with the INTROFF command.
INTR1:	›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 2 routine

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

EXAMPLES:

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

 INTR1:
 program statements
 RETURN
```

OR

Boolean Operator

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

PROGRAM SYNTAX: expression1 OR expression2

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

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

EXAMPLES:

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

OUT

I/O Operator

ACTION:

Sets or returns the condition of a specified digital output.

PROGRAM SYNTAX:

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

REMARKS:

n is the specified output (1-6).

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

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

The output are assigned as follows:

Output	Output Designation
1	OUT 1
2	OUT 2
3	BCD0/OUT3
4	BCD1/OUT4
5	BCD2/OUT5
6	BCD3/OUT6

EXAMPLES:

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

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

OUTLIMIT

ACTION:

Sets or returns the servo command voltage limit

PROGRAM SYNTAX:

OUTLIMIT=expression
OUTLIMIT used in an expression

REMARKS:

Limits the magnitude of the servo loop's output voltage. OUTLIMIT is set to 10 volts at power up. OUTLIMIT can be set between 0 and 10 volts inclusive. Setting it to a value outside this range will cause it to be set to the nearest valid value. Ex. OUTLIMIT = 5 will limit the servo output to +/- 5 volts. Reading OUTLIMIT returns the present setting in volts.

OUTLIMIT= expression
sets the OUTLIMIT to the expression value

OUTLIMIT
returns the current value of the OUTLIMIT.

EXAMPLES:

```
OUTLIMIT=5 ' sets analog output voltage limit to 5 volts.

x=OUTLIMIT  'sets variable x to the OUTLIMIT value.
```

PRINT

ACTION: Transmits designated data via the designated serial port.

PROGRAM SYNTAX:

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

REMARKS:

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

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

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

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

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

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

EXAMPLES:

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

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

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

PRINT USING

ACTION:

Prints strings character or formatted numbers.

PROGRAM SYNTAX:

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

REMARKS:

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

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

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

```
a$="Output \#1 is # which is the coolant control"
PRINT USING #1,a$,OUT(1)
    The resulting serial output:
    Output #1 is n which is the coolant control
    where: n is the state of output (1)
```

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

```
ACCEL=10000
DECEL=20000
a$="Acc= 000000   Dcc= 000000"
PRINT USING#1,a$,accel,decel
    The resulting serial output:
    Acc= 010000  Dcc= 020000
```

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

Example:

```
ABSPOS=1000.54
a$="Position= +0##.##"
PRINT USING #1,a$,abspos
        The resulting serial output:
        Position= +***.**
```

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

+ The sign of the number will always be printed. The default prints the negative sign and substitutes a space for the positive sign.

# Represents a digit position. If no data exists at the digit position substitutes a space. The digit field will always be filled.

0 Represents a digit position. If no data exists at the digit position substitutes a zero. The digit field will always be filled.

. A decimal point may be inserted at any position in the field.

The valid formats are:

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

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

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

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

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

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

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

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

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

EXAMPLES:

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

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

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

REGLIMIT

Over Travel Limit

ACTION:

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

PROGRAM SYNTAX:

REGLIMIT - used in an expression
REGLIMIT=expression

REMARKS:

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

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

EXAMPLES:

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

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

RIGHT$

String Manipulation

ACTION:

Returns the rightmost characters of a string.

PROGRAM SYNTAX:

string1$=RIGHT$(string2$,n)

REMARKS:

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

EXAMPLES:

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

SOFTLIM NEG

Over Travel Limit

ACTION:

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

PROGRAM SYNTAX:

SOFTLIMNEG=expression
SOFTLIMNEG - used in an expression

REMARKS:

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

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

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

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

SOFTLIMNEG=expression
Sets the absolute travel distance.

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

EXAMPLES:

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

SOFTLIM OFF

ACTION: Disables the software over travel limits.

PROGRAM SYNTAX: SOFTLIMOFF

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

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

EXAMPLES:

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

SOFTLIM ON

Over Travel Limit

ACTION:

Enables the software over travel limits.

PROGRAM SYNTAX:

SOFTLIMON

REMARKS:

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

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

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

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

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

EXAMPLES:

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

SOFTLIMPOS

Over Travel Limit

ACTION:

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

PROGRAM SYNTAX:

SOFTLIMPOS=expression
SOFTLIMPOS - used in an expression

REMARKS:

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

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

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

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

SOFTLIMPOS=expression
Sets the absolute travel distance.

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

EXAMPLES:

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

SPEED

Trajectory Parameters

ACTION:

Sets and returns the target velocity of the motor.

PROGRAM SYNTAX:

SPEED = expression
SPEED - used in an expression

REMARKS:

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

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

EXAMPLES:

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

STOP

Motion

ACTION:

Stops any motion with a controlled stop.

PROGRAM SYNTAX:

STOP

REMARKS:

Stop the motor using the programmed decel and velocity profile.

EXAMPLES:

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

STR$

String Manipulation

ACTION: Returns a string representation of a numeric expression.

PROGRAM SYNTAX: string1$=STR$(numeric_expression)

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

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

EXAMPLES:

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

STRING$

String Manipulation

ACTION:

PROGRAM SYNTAX:

REMARKS:

Returns a string of characters.

string1$=STRING$(num,code)

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

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

EXAMPLES:

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

TIMER

Time Functions

ACTION: Sets or returns the timer value.

PROGRAM SYNTAX:

```
TIMER=expression
TIMER - used in an expression
```

REMARKS: TIMER = expression

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

TIMER

Returns the current timer value to the variable.

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

EXAMPLES:

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

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

UCASE$

String Manipulation

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

PROGRAM SYNTAX:

```
string1$=UCASE$(string2$)
```

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

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

EXAMPLES:

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

UNITID

Daisy Chain

ACTION:

Returns the current ID.

PROGRAM SYNTAX:

UNITID - used in an expression

REMARKS:

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

EXAMPLES:

```
ID = UNITID                        'sets variable ID to the unit id number
IF VAL(unitid$) = ID THEN          ' if received unit id matches the unit id
                                   ' number execute the following statements
         YYY
         YYY
                                   END IF
```

VAL

String Manipulation

ACTION:

Return the numeric value of a string.

PROGRAM SYNTAX:

VAL(n$) - used in an expression

REMARKS:

n$ is the designated string.

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

EXAMPLES:

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

WAIT

Time Functions

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

PROGRAM SYNTAX: WAIT=expression

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

EXAMPLES:

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

WAITDONE

Motion

ACTION:

PROGRAM SYNTAX: Waits for a motion to be completed.

REMARKS: WAITDONE

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

An alternate way to accomplish the WAITDONE function is as follows:

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

EXAMPLES:

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

WNDGS

ACTION:

Enables or disables the servo drive.

PROGRAM SYNTAX:

WNDGS=expression
WNDGS - used in an expression

REMARKS:

The drive enable output powers up in the off state. This insures an initially safe condition. The WNDGS command controls the drive enable output. WNDGS =1 turns the enable on and WNDGS =0 turns it off.

Although the WNDGS command can be executed at any time, the drive enable output is only changed when motion is not occurring. The state of the drive enable output can be read using the "WNDGS" command. A return value of 0 = not enabled, 1 = enabled.

When the drive enable is off, the servo loop's integral term is zeroed and the servo loop output is 0v. When the drive enable is turned on, the commanded position (ABSPOS) is set equal to the encoder position (ENCPOS).This forces the position error to zero so that the servo loop output does not cause unexpected motion. If motion is commanded when the drive is disabled, error code 135 is set and no motion occurs.

The expression value must be zero or a positive number.

EXAMPLES:

```
PRINT#1,WNDGS     ' Prints the state of the servo drive windings
WNDGS=1           'enables the servo drive.
WNDGS=0           'disables the servo drive.
```

Section 6.2

Host Commands Reference Guide

6.2.0 Host Commands

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

6.2.1 Host Commands Grouped by Function

MOTION		**PAGE**
BUSY (BS)	Returns the motion status of the axis.	148
CMDPOS	Returns the commanded position of the motor in units.	149
EVENT1 (E1)	Sets enable or disable and trigger state or returns event1 input state.	155
EVENT2 (E2)	Sets enable or disable and trigger state or returns event2 input state.	155
FEEDHOLD (FH)	Control stops any motion.	156
JOG (J) **H**	Runs continuously in the specified direction.	160
MOVEA (MA) **H**	Initiates an absolute indexed move .	162
MOVEHOME (MH) **H**	Run until the home input is activated.	162
MOVEI (MI) **H**	Initiates an incremental indexed move .	163
MOVEREG (MR) **H**	Runs until the registration input is activated, then moves the specified distance.	163
STOP (S)	Control stop any motion.	169
WNDGS (WN)	Enable/disable servo drive.	170

TRAJECTORY PARAMETERS

ABSPOS (P) **H**	Sets or returns the absolute position.	147
ACCEL (AC) **H**	Sets or returns the acceleration rate in units/sec/sec.	147
DECEL (DC) **H**	Sets or returns the deceleration rate in units/sec/sec.	151
DIST **H**	Returns the distance moved from the start of the last commanded motion or changes the move distance during indexed motion.	152
ENCPOS (EP)	Returns the encoder absolute position.	152
ENCSPD (ES)	Returns the current speed.	153
SPEED (SPD)	Sets or returns the commanded target speed.	169

SERVO PARAMETERS

FOLERR (FE) **H**	Sets or returns the following error.	156
INTLIM (IL) **H**	Sets or returns the servo axis integral limit.	159
KD **H**	Sets or returns the servo axis derivative gain.	160
KI **H**	Sets or returns the servo axis integral gain.	160
KP **H**	Sets or returns the servo axis proportional gain.	161
KVFF **H**	Sets or returns the servo axis velocity feed forward gain.	161
OUTLIMIT (OL) **H**	Sets or returns the torque output voltage limit.	164

I/O		**PAGE**
ANALOG (AN)	Returns the analog input voltage.	147
BCD	Returns the BCD switches value.	148
IN (I)	Returns the discrete input state of the specified input.	159
OUT (O)	Sets the discrete output state of the specified output	164

TRAVEL LIMITS

HARDLIMOFF (HL0)	Disables hard limits.	158
HARDLIMON (HL1)	Enables hard limits.	158
REGLIMIT (RL) H	Sets or returns the MOVEREG limit distance.	165
SOFTLIMNEG (SLN) H	Sets or returns the software absolute negative travel distance.	167
SOFTLIMOFF (SL0)	Disable soft limits.	167
SOFTLIMON (SL1)	Enables soft limits	168
SOFTLIMPOS (SLP) H	Sets or returns the software absolute positive travel distance.	168

MISCELLANEOUS

CLRNVR	Initialize all location in NVR memory to 0.	149
DIR	Returns the user project information.	151
ERR	Return error number.	154
FREEMEM	Return the user program free memory byte value.	157
NVR	Returns or stores a REAL or INTEGER value to NVR memory.	163
RESET	Reset operating system.	165
REVISION (REV)	Returns operating system revision and date.	166
RUN H	Execute user program from the start.	166

IMMEDIATE

ABACKSPACE@	Delete one character in the host buffer.	148
ACTRL-A@	Stop motion immediately and terminate program execution.	150
ACTRL-C@	Stop motion immediately and terminate program execution. The analog output voltage for a servo is disabled.	150
AESCAPE@	Immediate Host command.	154

DAISY CHAINING

<nn	Enables a specific unit (nn = 01 to 32)	146
<nn?	Enables a specific unit (nn = 01 to 32) which then sends back its unit ID.	146
<00	Places all units on the chain in a command listen mode, the units can not transmit.	132

GEARING

ENCPOS2	Returns the Encoder 2 position in units**.**	152
ENCSPD2	Returns the current Encoder 2 velocity in units/second**.**	152

HIMPORTANT:

Host commands indicated with a (**H**) **WILL NOT** execute if a program is executing. The program execution must be halted, and the Host command re-entered for it to take effect.

6.2.2 Host Command Summary (alphabetical list)

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

		PAGE
<nn	Enables a specific unit (nn = 01 to 32)	146
<nn?	Enables a specific unit (nn = 01 to 32) which then sends back its unit ID.	146
<00	Places all units on the chain in a command listen mode, the units can not transmit.	146
A		
ABSPOS (P)	Sets or returns the absolute position.	147
ACCEL (AC)	Sets or returns the acceleration rate in units/sec/sec.	147
ANALOG (AN)	Returns the analog input voltage.	147
B		
BACKSPACE	Delete one character in the host buffer.	148
BCD	Returns the BCD switches value.	148
BUSY (BS)	Returns the motion status of the axis.	148
C		
CLRNVR	Initialize all location in NVR memory to 0.	149
CMDPOS	Returns the commanded position of the motor in units.	149
CTRL-A	Stop motion immediately and terminate program execution.	150
CTRL-C	Stop motion immediately and terminate program execution. The servo drive enable is turned off.	150
D		
DECEL (DC)	Sets or returns the deceleration rate in units/sec/sec.	151
DIR	returns the user project information.	151
DIST	Returns the distance moved from the start of the last commanded motion or changes the move distance during indexed motion.	152
E		
ENCPOS (EP)	Returns the encoder absolute position.	152
ENCPOS2	Returns the Encoder 2 position in units.	152
ENCSPD (ES)	Returns the current speed.	153
ENCSPD2	Returns the current Encoder 2 velocity in units/second.	153
ERR	Returns error number.	154
ESCAPE	Immediate host command.	154
EVENT1 (E1)	Sets enable or disable and trigger state of event1.	155
EVENT2 (E2)	Sets enable or disable and trigger state of event2.	155
F		
FEEDHOLD (FH)	Control stops any motion.	156
FOLERR (FE)	Sets or returns the following error.	156
FREEMEM	Return the user program free memory byte value.	157

ABSPOS

ACTION:

PROGRAM SYNTAX:

REMARKS:

6.2.3 Host Commands - Alphabetical Listing

Functional list of all HOST commands with syntax and examples

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

Daisy Chain

ACTION:

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

PROGRAM SYNTAX:

<nn cr

REMARKS:

Ann@ is a unit id number from 01,02...32. Leading zeros are required when specifying unit id numbers less than 10. 00 is a special case as described below.

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

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

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

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

EXAMPLES:

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

Trajectory Parameters

Sets or returns the commanded absolute position of the motor.

ABSPOS=number cr
ABSPOS cr

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

See Programming Command **ABSPOS.**

EXAMPLES:

ABSPOS=2 'sets absolute position to 2 units.

ABSPOS 'returns the current absolute position

ACCEL

Trajectory Parameters

ACTION:

on value of an axis.

PROGRAM SYNTAX:

REMARKS:

ed in place of ACCEL.

See Programming Command **ACCEL.**

EXAMPLES:

s the acceleration rate to 2 units/sec^2.

ANALOG

I/O Operator

ACTION:

e in volts.

PROGRAM SYNTAX:

REMARKS:

ed in place of ANALOG.

See Programming Command **ANALOG.**

EXAMPLES:

rns the analog input voltage value.

"BACK SPACE"

Immediate

ACTION:

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

PROGRAM SYNTAX

Press the BACKSPACE (²) key or send ASCII 08.

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

BCD

I/O Operator

ACTION:

Returns the value on the BCD switches.

PROGRAM SYNTAX:

BCD cr

REMARKS:

See Programming Command **BCD.**

EXAMPLES:

BCD switches value.

BUSY

Motion

ACTION:

PROGRAM SYNTAX:

REMARKS:

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

See Programming Command **BUSY**

EXAMPLES:

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

CLRNVR

NVR Memory

ACTION: Initialize all location in NVR memory to 0.

PROGRAM SYNTAX: CLRNVR cr

REMARKS: If an initialization value other than zero is required execute the following basic program.

```
REAL data
INTEGER count

data= 1.0                         ‘ initialization value
FOR count=1 to 400
                NVR(count)=data       ‘ initialize NVR location
NEXT count
END
```

EXAMPLES:

```
CLRNVR                            ‘ initializes all NVR location to 0
```

CMDPOS

Motion

ACTION: Returns the commanded position of the motor in units.

PROGRAM SYNTAX: CMDPOS cr

REMARKS: This command is defined in more details Section 8 Electronic Gearing.

"CTRL-A"

Immediate

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

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

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

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

"CTRL-C"

Immediate

ACTION: Performs the same function as the CTRL-A, but also de-energizes the servo system by turning off the DRIVE ENABLE signal and commanding 0V on the SERVO CMD analog output.

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

REMARKS: "CTRL C" will stop program execution and motion. When motion stops the servo drive is disabled.

To re-enable the servo drive use the WNDGS=1 command.

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

DECEL

Trajectory Parameters

ACTION: Sets or returns the deceleration value.

PROGRAM SYNTAX:

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

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

REMARKS: See Programming Command **DECEL.**

EXAMPLES:

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

DIR

Miscellaneous

ACTION: Returns the user project information.

PROGRAM SYNTAX: DIR cr

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

Returns the following ASCII format:

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

where:

n.nn project compiler version.

pppppppp project name. Up to 8 characters can be used for a project name. If less than 8 characters is used to identify a project the trailing characters will be spaces.

mm month the project was compiled.

dd day the project was compiled.

yyyy year the project was compiled.

hh hour the project was compiled.

mm minutes the project was compiled.

EXAMPLES:

```
DIR cr                ' with no project loaded
        crlf

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

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

DIST

Trajectory Parameters

ACTION:

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

PROGRAM SYNTAX:

DIST=number cr
DIST cr

REMARKS:

See Programming Command **DIST.**

EXAMPLES:

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

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

ENCPOS

Trajectory Parameters

ACTION:

Returns the encoder position.

PROGRAM SYNTAX:

ENCPOS cr

REMARKS:

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

See Programming Command **ENCPOS.**

EXAMPLES:

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

ENCPOS2

Gearing

ACTION:

Returns the Encoder 2 position in user units.

PROGRAM SYNTAX:

ENCPOS2 cr

REMARKS:

This command is defined in more detail in Section 8 Electronic Gearing.

ENCSPD

Trajectory Parameters

ACTION:

PROGRAM SYNTAX:

REMARKS:

Returns the current encoder speed in units/second.

ENCSPD cr

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

See Programming Command **ENCSPD.**

EXAMPLES:

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

ENCSPD2

Gearing

ACTION:

PROGRAM SYNTAX:

REMARKS:

Returns the current Encoder 2 velocity in units/second.

ENCSPD2 cr

This command is defined in more details in the Gearing section of the manual.

ERR

Miscellaneous

ACTION: Returns the error status of the controller.

PROGRAM SYNTAX: ERR cr

REMARKS: See Programming Command **ERR.**

EXAMPLES:

ERR cr ' return the error status.

"ESCAPE"

Immediate

ACTION:

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

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

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

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

EXAMPLES:

```
"ESCAPE" ABSPOS cr          ' returns the absolute position

"ESCAPE" STOP cr            ' stops motion

 "ESCAPE" FEEDHOLD cr       ' Feedhold motor
```

EVENT1

Motion

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

PROGRAM SYNTAX: EVENT1=number cr

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

See Programming Command **EVENT1**

EXAMPLES:

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

EVENT2

Motion

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

PROGRAM SYNTAX: EVENT2=number cr

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

See Programming Command **EVENT2**

EXAMPLES:

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

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

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

FEEDHOLD

Motion

ACTION:

This command stops all motion by decelerating at the programmed decel rate. The user program continues to run. Motion may be continued by issuing a RUN host command or toggling the RUN hardware input from inactive to active.

PROGRAM SYNTAX:

FEEDHOLD cr

ABBREVIATION:

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

REMARKS:

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

To resume the stopped motion issue a Arun" host command.

EXAMPLES:

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

FEEDHOLD cr

FOLERR

Servo Parameters

ACTION:

PROGRAM SYNTAX:

Sets or returns the position error limit. When the position error limit is exceeded, any ongoing motion is stopped and an error is set.

ABBREVIATION:

FOLERR=number cr
FOLERR cr

REMARKS:

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

See Programming Command **FOLERR.**

EXAMPLES: sition error limit is set to .5 units

FREEMEM

ACTION:

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

PROGRAM SYNTAX:

FREEMEM cr

REMARKS:

The return format is:

```
        tttt,nnnn
        where: tttt       total number of 8 bit bytes available.
                 nnnn     number of 8 bit byte remaining.
```

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

EXAMPLES:

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

HARDLIM OFF

Travel Limits

ACTION: Disables the hardware limit inputs.

PROGRAM SYNTAX: HARDLIMOFF cr

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

REMARKS: See Programming Command **HARDLIMOFF.**

EXAMPLES:

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

HARDLIM ON

Travel Limits

ACTION: Enables the hardware limit inputs.

PROGRAM SYNTAX: HARDLIMON cr

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

REMARKS: See Programming Command **HARDLIMON.**

EXAMPLES:

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

IN

I/O Operator

ACTION: Returns the state of a digital input.

PROGRAM SYNTAX: IN(nn) cr

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

REMARKS: See Programming Command **IN.**

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

INTLIM

Servo Parameters

ACTION: Sets the Integral limit for the controller. This is the limit of the contribution to the servo output from the integral of the position error.

PROGRAM SYNTAX:

INTLIM=number cr
INTLIM cr

ABBREVIATION: Abbreviation **IL** can be used in place of INTLIM.

REMARKS: See Programming Command **INTLIM.**

EXAMPLES: INTLIM=5 ' Sets the integral limit to 5 volts.

JOG

Motion

ACTION: Move in the specified direction.

PROGRAM SYNTAX: JOG = number cr

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

REMARKS: See Programming Command **JOG.**

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

KD

Servo Parameters

ACTION: Sets or returns the derivative gain for the servo motor.

PROGRAM SYNTAX:

KD=number cr
KD cr

REMARKS: See Programming Command **KD.**

EXAMPLES: ivative gain to 10 milliseconds.

KI

Servo Parameters

ACTION:

PROGRAM SYNTAX: ain of the servo motor.

REMARKS:

See Programming Command **KI.**

EXAMPLES: KI=1 'Sets the integral gain to 1 millisecond.

KI ' returns the current value for integral gain

KP

ACTION:

Sets or returns the proportional gain of the servo motor.

PROGRAM SYNTAX:

KP=number cr
KP cr

REMARKS:

See Programming Command **KP.**

EXAMPLES:

KP(2)=50 'Sets the proportional gain to 50 millivolts/encoder count (.05 volts/encoder count).

KP ' returns the current value for proportional gain

KVFF

ACTION:

feed forward gain value for the servo motor.

PROGRAM SYNTAX:

KVFF cr

REMARKS:

See Programming Command **KVFF.**

EXAMPLES:

s the velocity feed forward gain to 100%.

KVFF ' return the current velocity feed forward gain

MOVEA

Motion

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

PROGRAM SYNTAX: MOVEA = number cr

ABBREVIATION: Abbreviation **MA** can be used in place of MOVEA.

REMARKS: See Programming Command **MOVEA.**

EXAMPLES:

```
MOVEA=2.5              '  moves to absolute position of 2.5 units.
```

MOVE HOME

Motion

ACTION:

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

PROGRAM SYNTAX: MOVEHOME = number cr

ABBREVIATION: Abbreviation **MH** can be used in place of MOVEHOME.

REMARKS: See Programming Command **MOVEHOME.**

EXAMPLES:

```
MOVEHOME=+1      'Returns to mechanical home in the Positive direction
```

MOVEI

Motion

ACTION: Initiates an incremental move.

PROGRAM SYNTAX: MOVEI = number cr

ABBREVIATION: Abbreviation **MI** can be used in place of MOVEI.

REMARKS: See Programming Command **MOVEI.**

EXAMPLES:

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

MOVEREG

Motion

ACTION: Runs the motor until the mark registration input is activated, then moves the motor the desired registration distance without stopping.

PROGRAM SYNTAX: MOVEREG =number cr

ABBREVIATION: Abbreviation **MR** can be used in place of MOVEREG.

REMARKS: See Programming Command **MOVEREG.**

EXAMPLES: MOVEREG=+2.5   'Initiates a registration cycle in the positive direction with a move of 2.5 units after the mark registration input is activated.

NVR

NVR Memory

ACTION: Returns or stores a REAL or INTEGER value to NVR memory.

PROGRAM SYNTAX:

NVR(element) cr
NVR(element) = number cr

REMARKS: See programming command **NVR**.

EXAMPLES:

```
NVR(400)=34.5          ‘ store 34.5 in NVR location 400
NVR(400)               ‘ returns 34.5 from NVR location 400
```

OUT

I/O Operator

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

PROGRAM SYNTAX:

```
OUT(n) = number cr
OUT(n) cr
```

ABBREVIATION: Abbreviation **O** can be used in place of OUT.

REMARKS: See Programming Command **OUT.**

EXAMPLES:

```
OUT(1)=1            ' sets OUT 1 to the active state.

OUT(2)=0            ' sets OUT 2 to the inactive state
```

OUTLIMIT

Servo Parameters

ACTION: Sets or returns the servo command voltage limit.

PROGRAM SYNTAX:

```
OUTLIMIT=number cr
OUTLIMIT cr
```

ABBREVIATION: Abbreviation **OL** can be used in place of OUTLIMIT.

REMARKS: See Programming Command **OUTLIMIT.**

EXAMPLES:

```
OUTLIMIT=5 ' sets analog output voltage limit to 5 volts.

OUTLIMIT            ' returns the current analog output limit.
```

REGLIMIT

Travel Limits

ACTION: Sets or returns the maximum mark registration distance before indicating an error and stopping motion.

PROGRAM SYNTAX:

REGLIMIT cr
REGLIMIT=number cr

ABBREVIATION: Abbreviation **RL** can be used in place of REGLIMIT.

REMARKS: See Programming Command **REGLIMIT.**

EXAMPLES:

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

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

REGLIMIT              ' returns the current REGLIMIT value
```

RESET

Miscellaneous

ACTION: Resets the system.

PROGRAM SYNTAX: RESET cr

REMARKS: This command causes the system to halt, and then restart as though power had been cycled.

REVISION

Miscellaneous

ACTION: Returns the current revision level of the controller's operating system software.

PROGRAM SYNTAX: REVISION cr

ABBREVIATION: Abbreviation **REV** can be used in place of REVISION.

REMARKS: The return format for this command is:
TDC REV n.nn mm/dd/yy
where:

n.nn	TDC software revision
mm	month
dd	day
yy	year

RUN

Miscellaneous

ACTION: Starts the user program or resumes from a FEEDHOLD condition which has been generated either by the hardware input **Feedhold** or by the **FEEDHOLD** host command.

PROGRAM SYNTAX: RUN cr

REMARKS: Starts execution of the user program if a Feedhold condition does not exist.

Resumes motion if a Feedhold condition exists.

EXAMPLES:

RUN

SOFTLIM NEG

Travel Limits

ACTION:

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

PROGRAM SYNTAX:

SOFTLIMNEG=number cr
SOFTLIMNEG cr

ABBREVIATION:

Abbreviation **SLN** can be used in place of SOFTLIMNEG.

REMARKS:

See Programming Command **SOFTLIMNEG.**

EXAMPLES:

```
SOFTLIMNEG= -4    ' sets the absolute software travel limit to -4 units.

SOFTLIMNEG        ' returns the software travel limit value
```

SOFTLIM OFF

Travel Limits

ACTION:

Disables the software over travel limits.

PROGRAM SYNTAX:

SOFTLIMOFF cr

ABBREVIATION:

Abbreviation **SL0** can be used in place of SOFTLIMOFF.

REMARKS:

See Programming Command **SOFTLIMOFF.**

EXAMPLES:

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

SOFTLIM ON

Travel Limits

ACTION: Enables the software over travel limits.

PROGRAM SYNTAX: SOFTLIMON cr

ABBREVIATION: Abbreviation **SL1** can be used in place of SOFTLIMON.

REMARKS: See Programming Command **SOFTLIMON.**

EXAMPLES:

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

SOFTLIM POS

Travel Limits

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

PROGRAM SYNTAX:

SOFTLIMPOS=number cr
SOFTLIMPOS cr

ABBREVIATION: Abbreviation **SLP** can be used in place of SOFTLIMPOS.

REMARKS: See Programming Command **SOFTLIMPOS.**

EXAMPLES:

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

SOFTLIMPOS          ' returns the current SOFTLIMPOS value.
```

SPEED

Trajectory Parameters

ACTION: Sets and returns the target velocity of the motor.

PROGRAM SYNTAX:

SPEED = number cr
SPEED cr

ABBREVIATION: Abbreviation **SPD** can be used in place of SPEED.

REMARKS: See Programming Command **SPEED.**

EXAMPLES:

```
SPEED=2.0          'sets target velocity to 2 units/sec.

SPEED              ' returns 2.0
```

STOP

Motion

ACTION: Stops any motion with a controlled stop.

PROGRAM SYNTAX: STOP cr

ABBREVIATION: Abbreviation **S** can be used in place of STOP.

REMARKS: Stop the motor using the programmed decel and velocity profile.

NOTE: If a program is executing, STOP will stop the present move. However, program execution will continue and subsequent MOVE commands within the program will execute. To stop motion immediately and terminate program execution see Cntl-A and Cntl-C.

EXAMPLES:

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

WNDGS

ACTION:

Enables or disables the servo drive.

PROGRAM SYNTAX:

WNDGS=number cr

ABBREVIATION:

Abbreviation **WN** can be used in place of WNDGS.

REMARKS:

See Programming Command **WNDGS.**

EXAMPLES:

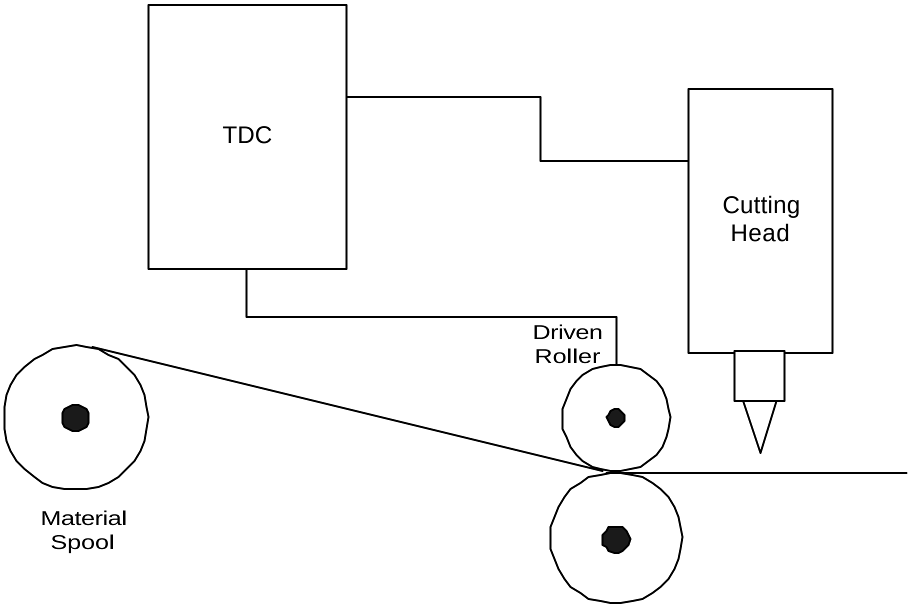
```
WNDGS=0           'disables the servo drive.

WNDGS=1           'enables the servo drive.
```

Section 7

Programming Examples

Cut to Length Application



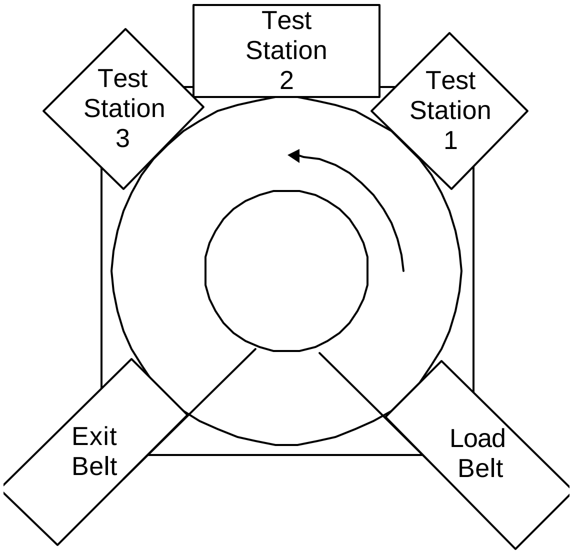
The application requires a servo motor to run a pair of nip rollers that draw material from a spool. This material could be anything from paper to steel. The requirements for this application are:

Wait for activation of input 1 from an external device. This device may be an operator input or PLC.
Feed out a length of material. For this example, 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 .25 seconds to allow the blade to return home.
Repeat the process if input 2 is not activated.

<u>Program Code</u>:

```
begin:                                      ›Label for program return .
        do : loop until (in(1)=1)           ›Wait for input 1 to become active.
        movei=12                            ›Move 12 inches (incremental move).
        waitdone                            'wait for motion to be completed
        out(1)=1                            ›Turn on output 1, cutting blade activation.
        wait=1                              ›Wait for 1 second.  Wait for cut to happen.
        out(1)=0                            ›Turn off output 1, cutting blade deactivation.
        wait=.25                            ›Wait for cutting blade to return, .25 sec.
        if in(2)=0 then goto begin          ›Return to beginning of program.
end                                         ›End of program.
```

Rotary Table Application, Test Stations



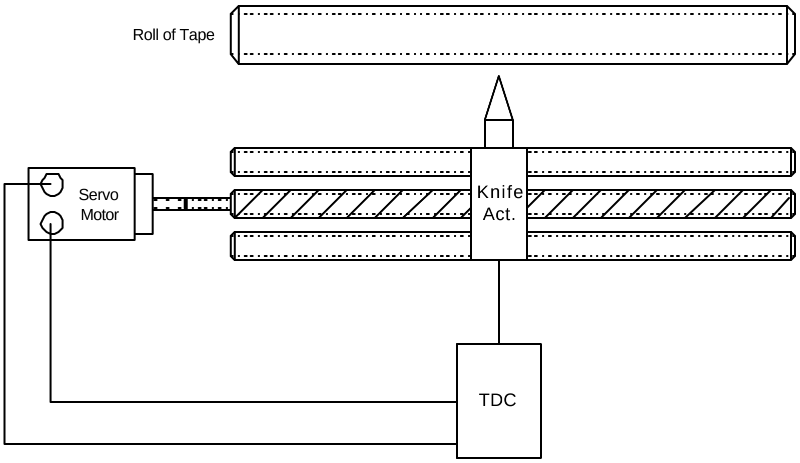
In this application, a customer needed to load a part onto a rotary table via the load belt. Once there, the part must be tested at three stations. The application requires:

A sensor to tell the table to jog until a sensor before the first station and index 45E
Turn on an output to tell the belt to stop until testing is complete at all the stations.
The test procedure requires the sample to be at each station until an input is activated on the control. Each test station is 45E apart.
After the last test station, the part is rotated 90E to the exit station where it is carried out. A sensor will then tell the controller to start the load belt for the next part.

Program Code (Rotary Table):

```
begin:                                     › Label to start program
  do : loop until(in(1)=1)                 › Wait until a part is detected by the sensor
     out(2)=1                              › Turn off the exit belt motor
     out(1)=1                              › Turn off the loading belt motor
     movei=+90                             › Move table 90E
     waitdone                              ' Wait for motion to be completed
     do: loop until (in(2)=1)       › Wait for test station to complete testing and turn on input 2.
     count =0                              › Initialize a counter to zero
     do                                    › Do loop begin
            movei=+45                      › Move rotary table +45E to the next station
            waitdone                       › Wait until the motor stops.
            do : loop  until(in(3)=1 or in(4)=0) ›Stay in this loop until testing at stations 2 & 3 are complete.
            count=count+1                  › Increment counter by 1.
     loop until count=2                    › Do loop end.
     movei=+90                             › Move the part 90E to the exit belt
     waitdone                              ' Wait for motion to be completed
     if in(5)=0 then                       › Check input 5 if it is inactive if not continue if active end.
            out(2)=0                       › Turn on exit belt.
            do : loop until in(6)=1        › Stay in loop until the sensor is activated.
            out(1)=0                       › Turn on loading belt.
            goto begin                     › Return to the beginning of the program
     end if                                › End of the if statement.
end                                        › End of the program.
```

Slitting Machine Application



A manufacturer of adhesive tape uses a machine that takes a wide roll of tape and slits then to the correct sizes. The program must be written to make tape sizes of 2", 1" and 0.5" from a 10' roll. The size of the tape will be determined by the inputs selected. The machine will operate as follows:
Return to mechanical home.
If input 1 make 0.5" size of tape.
If input 2 make 1.0" size of tape.
If input 3 make 2.0" size of tape.
Loop until a selection is made and input 4 is active
If input 5 is active end program else return to electrical home
Restart program.

Program Code (Slitting machine):

```
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.
```

Section 8

Electronic Gearing

8.1 - Gearing Description

Gearing allows the controller to follow a selected master velocity source at a specified ratio. The master source can be an external velocity source or a program command source.

The **GEAREXT** or **GEARINT** programming command determines the selection. A quadrature encoder is used as the external source, program command **GEARVEL** is used as the internal source. The external source velocity can also be scaled using the **GEARRATIO** command.

The default is external velocity source generated by the encoder 2 quadrature inputs with a **GEARRATIO** of 1.

Note: For true gearing operation the master starting velocity should be zero and the gearing rate limit should be greater than or equal to the master acceleration rate times the GEARRATIO.

8.2 - Gearing Features

- Simple Basic commands.
- Basic command selectable Master source, **GEAREXT** and **GEARINT** commands.
- Master internal source can be a variable or an expression and is specified in units/second, **GEARVEL** command.
- Motor can be ratioed from an external master source, **GEARRATIO** command. The ratio can be changed in small increments during gearing motion.
- The master velocity source can be monitored using the **ENCSPD2** command.
- The scaled master position can be monitored using the **ENCPOS2** command.
- Soft start, an acceleration rate limit is imposed from a sudden start. The **ACCEL** rate is the rate limit. Excess counts will be backlogged.
- Gearing Hard limits and Soft limits can be enabled. If the limit condition is met it produces the same errors as normal motion.
- **MOVEA**, **MOVEI** and **JOG** motion can be superimposed on a gearing motion.
- A short excessive rate change can be tolerated without positioning error.

8.3 – Enabling/Disabling Gearing

The gearing mode can be enabled or disabled by issuing program commands. The **GEARON** command enables gearing while the **GEAROFF** command disables gearing. The default is gearing disabled.

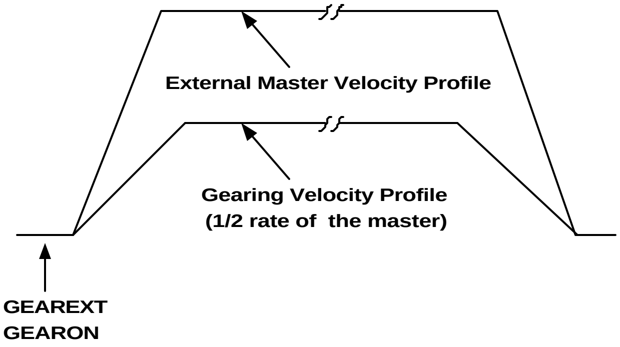
Note: When GEAROFF is commanded the motor will decelerate to a stop at the ACCEL rate selected.

8.4 – Gearing Motion

Gearing motion using an external encoder is allowed when the **GEAREXT** and **GEARON** command are encountered during program execution. The **GEARRATIO** will determine how the motor will follow the encoder velocity.

The following diagram depicts a typical gearing cycle.

Gearing Profile



8.5 – Velocity Rate Limit

A rate limit is imposed on the gearing velocity. The rate limit is determined by the value of the **ACCEL** command.

The default **ACCEL** is set to 50% of **the Max Accel** value when program execution begins if **ACCEL** has not been specified. This rate can be changed using the **ACCEL** program command during program execution.

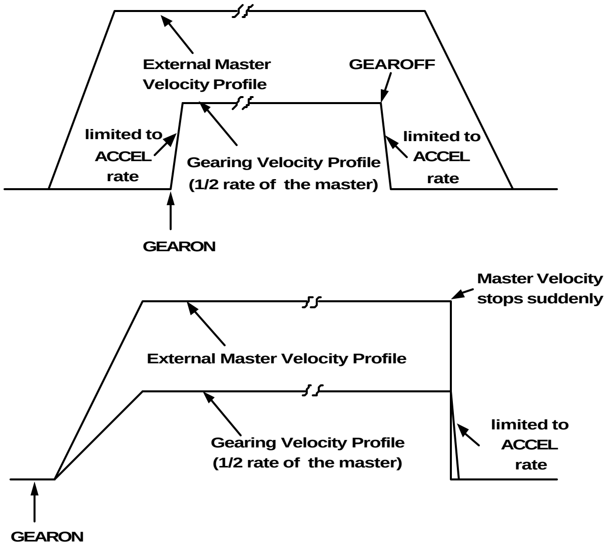
See Section 8.6 Gearing Anomalies for more details.

8.6 – Gearing Anomalies

Gearing Anomalies occur when gearing is enabled or disabled and the master velocity is greater than zero. The motor velocity limit change during these anomalies is controlled by the current **ACCEL** value.

The following diagrams depict these gearing anomalies.

Gearing Anomalies



8.7 – Advance / Recede Motion

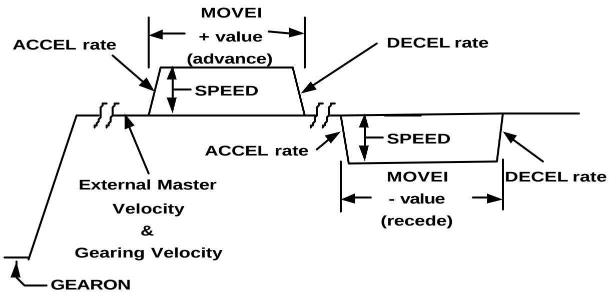
A commanded motion can be superimposed on a gearing motion. This is accomplished by issuing a JOG, MOVEI or MOVEA command while gearing motion is taking place.

Note: The MOVEHOME or MOVEREG commands should not be used during gearing motion.

The motion profile in which the MOVEI, MOVEA, and JOG commands will follow is based upon the current settings of SPEED and Accel/DECEL for the axis.

The following diagram depicts an advance and recede cycle.

Advance/Recede cycle



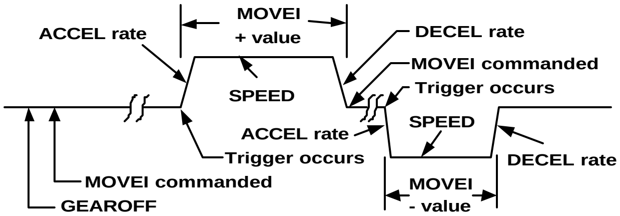
8.8 – Triggered Motion

Another feature of the controller is the ability to start a motion on a trigger event. The **MOTTRIG** command selects the trigger mode of operation for a **MOVEI**, **MOVEA**, **MOVEHOME**, **JOG** and **MOVEHOME** cycle.

The selections are no trigger, event 1 active, event 1 inactive, event 2 active and event 2 inactive.

The following diagram depicts a triggered move cycle.

Triggered Move cycle



8.9 – Gearing Command Listing

CMDPOS

ACTION:

Returns the commanded position of the motor in units.

PROGRAM SYNTAX:

CMDPOS – used in an expression

REMARKS:

CMDPOS is the motor commanded position at any time. The value returned is in user units.

If gearing is not used, **CMDPOS** will equal the normal commanded position, **ABSPOS**. When gearing mode is enabled, the motor's commanded position, **CMDPOS**, is the sum of the normal position, **ABSPOS**, and the scaled gearing position **ENCPOS2**. This allows the superposition of gearing motion and indexed motion to create advance or recede cycles relative to the master input. Note here that **ABSPOS** will represent the position command as a result of a **MOVEA**, **MOVEI** or **JOG** command, and will not account for changes in position due to gearing. Therefore **ABSPOS** can be used to determine how far the motor (follower) has advanced or receded **relative to the master**.

ENCPOS is the actual motor encoder position, which should track the CMDPOS. The **ABSPOS** is the normal motion position and the **ENCPOS2** is the Gearing position.

Whenever **ABSPOS** is set, the motor commanded position, **CMDPOS**, is set to the same value. **ENCPOS2** is set to zero.

EXAMPLES:

```
REAL pos

pos = CMDPOS          ‘ sets real variable pos equal to the current motor
                        commanded position.
```

ENCPOS2

ACTION: Returns the Encoder 2 position in user units.

PROGRAM SYNTAX: ENCPOS2 – used in an expression

REMARKS: Evaluates and returns the present encoder 2 position. In gearing mode the Encoder 2 position is the gearing master position.

The range of ENCPOS2 is ± 2,147,483,647 counts. If the range is exceeded the sign is reversed and counting continues in the same direction.

The encoder line counts in the configuration folder and the **GEARRATIO** are used to scale **ENCPOS2** into user units. Therefore, reading **ENCPOS2** returns the scaled encoder 2 position in user units (for example motor revolutions).

ENCPOS2 is initialized to 0 at **power-up**, and whenever an **ABSPOS = expression is commanded**, a **user program is downloaded** or a **RUN is commanded**.

EXAMPLES:

```
REAL y

y = ENCPOS2        ‘ sets real variable y equal to the current encoder
                     2 position.
```

ENCSPD2

ACTION: Returns the current Encoder 2 velocity in units/second.

PROGRAM SYNTAX: ENCSPD2 – used in an expression

REMARKS: Evaluates and returns the current encoder 2 velocity, gearing velocity.

The returned velocity value is a signed value and indicates the direction of the velocity. **ENCSPD2** uses the encoder line counts in the Configuration folder and the **GEARRATIO** to provide the velocity in user units/second (for example if user units/rev = 1, then **ENCSPD2** is in motor revolutions per second).

EXAMPLES:

```
REAL y

y = ENCSPD            ‘ sets real variable y equal to the current
                        encoder 2 velocity in units/second.
```

GEAREXT

ACTION: This command selects the encoder 2 external input port as the master velocity source for gearing.

PROGRAM SYNTAX: GEAREXT

REMARKS: The motor velocity tracks the external master encoder velocity when gearing is enabled. The ratio of the motor (follower) velocity to the master velocity is set using the **GEARRATIO** command

If the master velocity is a non-zero value when gearing is enabled the motor acceleration rate will be limited to the programmed **ACCEL** rate.

The **GEARON** and **GEAROFF** commands enable or disable tracking of the master velocity.

EXAMPLES:

```
GEAREXT     ‘ select the External velocity source for gearing
```

GEARINT

ACTION: This command selects the internal master velocity source for gearing. The value of the internal master velocity source is set using the **GEARVEL** command.

PROGRAM SYNTAX: GEARINT

REMARKS: The motor velocity tracks the **GEARVEL** commanded master velocity when gearing is enabled using the **GEARON** command.

If the master velocity is a non-zero value when gearing is enabled the motor acceleration rate will be limited to the programmed **ACCEL** rate.

The **GEARON** and **GEAROFF** commands enable or disable tracking of the master velocity.

EXAMPLES:

```
GEARINT      ‘selects the GEARVEL command as the velocity
              source  for gearing.
```

GEARON

ACTION: This command enables master velocity tracking in the gearing mode of operation.

PROGRAM SYNTAX:

```
GEARON
```

REMARKS: The **GEAREXT** and **GEARINT** commands select the master velocity source. The motor (follower) velocity will track the master velocity based on the **GEARRATIO** which must be properly set prior to executing the **GEARON** command. The **GEARRATIO** command may also be changed on the fly.

If the master velocity is a non-zero value when gearing is enabled the motor acceleration rate will be limited to the programmed ACCEL rate.

EXAMPLES:

```
GEARON      ‘ enables the master velocity for gearing
```

GEAROFF

ACTION: This command disables master velocity tracking in the gearing mode of operation.

PROGRAM SYNTAX:

```
GEAROFF
```

REMARKS: If the master velocity is a non-zero value when gearing is disabled the motor deceleration rate will be limited to the programmed ACCEL rate.

The GEAROFF condition is selected whenever a Gearing error occurs or when program execution begins.

EXAMPLES:

```
GEAROFF     ‘ disables the master velocity for gearing
```

GEAR RATIO

ACTION:

This command selects the motor (follower) to master encoder gearing ratio.

PROGRAM SYNTAX:

GEARRATIO = expression
GEARRATIO – used in an expression

REMARKS:

The expression sets the external velocity gear ratio, **motor revolutions / encoder 2 revolutions**. An expression value of 1.0 specifies a Gearing Ratio of 1. The expression value must be a positive number.

The gear ratio is only valid when the external velocity source (encoder 2) is selected as the master velocity using the **GEAREXT** command. External velocity source is the default selection when program execution begins.

The default **GEARRATIO** is 1.0 when a user program begins.

The Gear Ratio range **$5.96e^{-8}$ to 16.0** in $5.96\ e^{-8}$ increments. However, the practical range for gearing is .00025 to 10.

The Gear ratio may be changed during Gearing motion in small increments. Large changes may result in jerky motion and possibly a following error condition (see FOLERR).

EXAMPLES:

```
GEARRATIO=1.5          ‘ sets the gear ratio of motor/master to 1.5

REAL ratio
ratio = GEARRATIO      ‘sets real variable ratio equal to the current
                        GEARRATIO value.

‘ ramp GEARRATIO example
‘ changes GEARRATIO from 1.5 to .5 in 500 milliseconds
REAL          increment
INTEGER       x, count
GEARRATIO=1.5          ‘ sets the gear ratio of motor/master to 1.5
GEARON                  ‘ enable gearing
….                      ‘ program statements
count = 100             ‘ number of increments
increment = .01         ‘ increment size for GEARRATIO change
FOR x = 1 to count
        GEARRATIO = GEARRATIO - increment
        Wait = .005     ‘ time between increments
NEXT x
```

GEARVEL

ACTION:

Sets or returns the master velocity for internal gearing in units/second.

PROGRAM SYNTAX:

GEARVEL = expression
GEARVEL – used in an expression

REMARKS:

When the gearing master source is selected as internal using the **GEARINT** command, the **GEARVEL** expression specifies the master velocity in units/second. This velocity is NOT scaled by the **GEARRATIO**, which is only valid for an external velocity source.

The motor velocity tracks this velocity directly. However, the motor acceleration and deceleration rates are limited to the programmed **ACCEL** rate.

The **GEARON** and **GEAROFF** commands enable and disable tracking of the **GEARVEL** velocity in the **GEARINT** mode of operation.

If **GEAREXT** is set (default), then **GEARVEL** has no effect.

EXAMPLES:

```
GEAREXT                  ‘ External master velocity selected
GEARON                   ‘ Enable gearing using external velocity
….                       ‘ program statements
GEARVEL = ENCSPD2         ‘ save last external master velocity.
    GEARINT                  ‘ continue running at last external
                               master velocity.


    REAL gearspd
    gearspd = GEARVEL        ‘ sets real variable gearspd to the
                               current GEARVEL value.
```

8.10 – Cut to Length Application

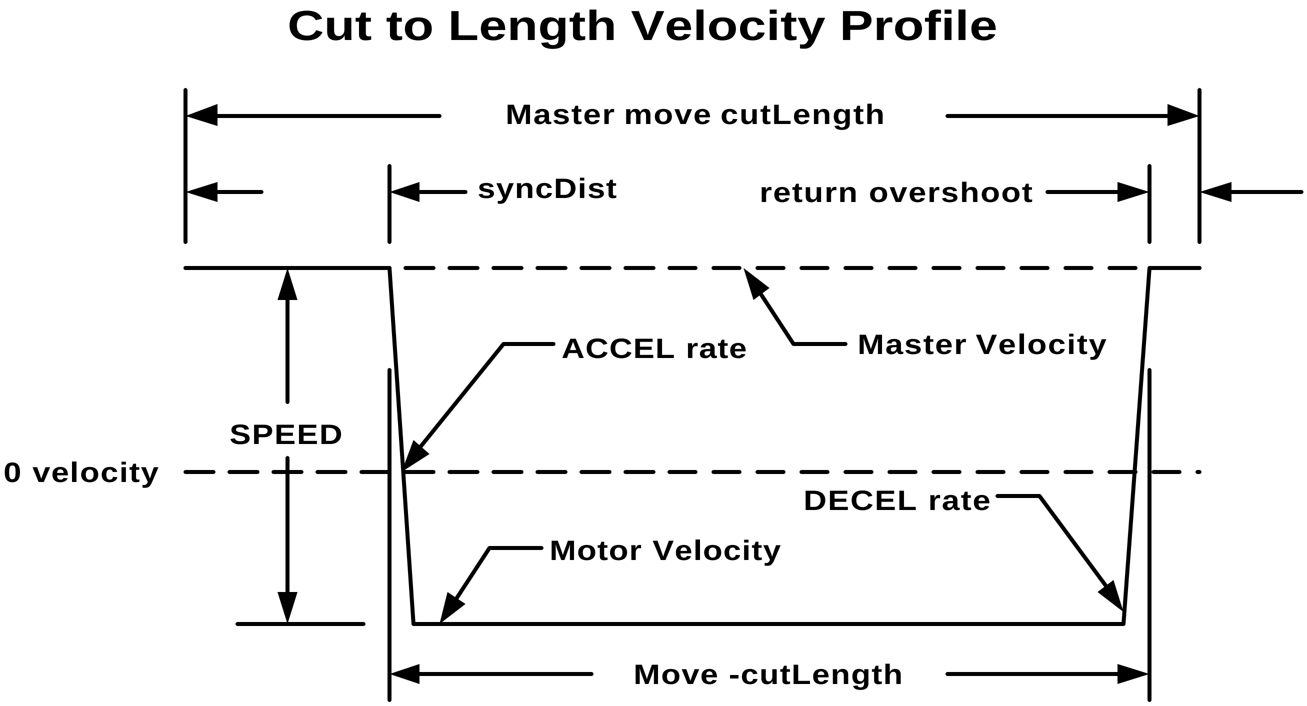
This application requires that material, being driven by an encoder source, be cut to a specific length while the material is moving. To accomplish this the cutting axis must run in synchronization with the material during the cutting cycle. In this example the cutter is engaged when OUT1=1 and removed when OUT1=0.

In the program listing below, the material cutting length is specified using variable **cutLength**. The distance the cutter is in synchronized with the material is specified using variable **syncDist**. The master velocity is derived from a quadrature Encoder signal source that is applied to the Encoder 2 port. The **Encoder 2 Line Count** value is used to specify the source value in the user program configuration.

The cutting cycle consists of engaging the cutter and waiting for the syncDist - .1 units to be traveled. The cutter is now disengaged and the remainder of the specified syncDist is traveled. A cutlength index motion traveling in the opposite direction is commanded. When this index motion is completed the cutter has overshot the next cutting position. The speedFactor is now adjusted to reduce the overshoot value of the next cycle. When the cutting position is reached the cutting cycle restarts.

Maximum cut error is ± (master velocity/500) units.

The velocity profile for the cut to length application is depicted below.



Cut to Length Application

Program Code (cuttolen.prj included):

```
'********************************************************************************
'** 1) Units/Rev =1 for this program
'** 2) External master source is assumed to be going in the positive direction for this program.
'** 3) Test code is used to simulate the application. It is not intended to measure cut accuracy.
'**    Comment this code out when running real parts.
'********************************************************************************

REAL           cutLength                    ' master cutting length
REAL           syncDist                     ' master cutting cycle distance traveled
REAL           velocity                     ' move velocity before being scaled
REAL           speedFactor                  ' velocity scale factor
REAL           initFactor                   ' initial scale factor
REAL           adjust                       ' move distance
REAL           compare                      ' targeted overshoot value
REAL           pos                          ' temporary variable
REAL           min,max,value,compValue      ' test only

‘************************** define cutting parameters ***************************
cutLength=10                                ' master cut length
syncDist=2                                  ' master cutting cycle distance traveled

‘************************ initialize system parameters ***************************
ABSPOS=0                                    ' cutting commanded starting position
FOLERR=2                                    ‘ servo position error limit =2 units
WNDGS=1                                     ‘ enable servo motor
GEARRATIO=1                                 ' gear ratio motor/master
ACCEL=500                                   ' rate limit and move acceleration rate
DECEL=ACCEL                                 ' move deceleration rate
GEAREXT                                     ' encoder 2 master source selected
velocity=1                                  ' initial move velocity will be modified if exceeded
speedFactor= 1/((cutLength - syncDist) / cutLength)
speedFactor = speedFactor + .5              ‘move velocity scale factor
initFactor=speedFactor                      ' initial move velocity scale factor
adjust = cutLength * GEARRATIO              ' move distance
syncDist=syncDist * GEARRATIO               ' motor cutting cycle distance
compare= -.5 * GEARRATIO                    ' target overshoot value

'******************************** test code ***********************************
min=adjust                                  'initialize min value
max=0                                       'initialize max value
compvalue =2  * adjust                      'initialize compare value
'********************************************************************************

‘************************** enable cutting cycle ********************************
GEARON                                      ' enable gearing, motion begins

'****************************** test code **************************************
PRINT#1,"press any key to stop program and print max/min results"
'********************************************************************************
```

```
DO
        ‘ ********************** cut cycle ******************************
        OUT(1)=1                                ' engage cutter
        DO                                      ' wait for distance
        LOOP UNTIL CMDPOS >= (syncDist - .1)
        OUT(1)=0                                ' disengage cutter
        DO                                      ' wait for sync distance to complete
        LOOP UNTIL CMDPOS >= syncDist

        ‘********************** recede cycle ****************************
        IF ENCSPD2 > velocity THEN              ' master velocity increased
                velocity=ENCSPD2                ' init new master velocity
                speedFactor=initFactor          ' set initial scale factor
        END IF
        SPEED=velocity * speedFactor            ' move target speed
        MOVEI = - adjust                        ' recede the cut length
        WAITDONE

        ‘******************** optimize recede speed ***********************
        pos=CMDPOS

        '************************ test code ******************************
        PRINT#1,pos                             ' print overshoot number

        '****************** adjust speed factor if required******************
        IF pos < compare THEN                   ' auto adjust return overshoot
            speedFactor =speedFactor * .95      ' adjust to get closer
        ELSE
            IF pos >= 0 THEN                    ' auto adjust return too short
                speedFactor = speedFactor * 1.5
            END IF
        END IF

        ‘****************** wait for cutting start position ********************
        DO
        LOOP UNTIL CMDPOS >= 0                  'wait for cutting starting position

'****************************** test code ********************************
        pos=ENCPOS2                             ' read input position
        value=compValue-pos                     ' calculated cut distance
        IF value > max THEN
                max=value                       ' maximum cut value
        END IF
        IF value < min THEN
                min=value                       ' minimum cut value
        END IF
        compValue= compValue + adjust           'next compare position

LOOP UNTIL INCHAR(1) >0                         ' stop the cycle request?
GEAROFF                                         ' disable gearing

'****************************** test code *********************************
PRINT#1,"maximum=";max / GEARRATIO,"minimum=";min / GEARRATIO

DO                                              ' wait for ramp down
LOOP UNTIL ENCSPD2=0
END
```

8.11 – Electronic Gear Box Application

This application requires a ratio between the master velocity and the servo motor. This example simulates a gearbox with a 5:1 reduction. The master source is a quadrature Encoder input.

Program Example (gearbox.prj included):

```
'***************** Gear box application program*****************

ACCEL = 50                    ‘ input rate limit
ABSPOS=0                      ‘ set Abspos=0 and Encpos2=0
FOLERR =2                     ‘ servo position error limit = 2 units
WNDGS=1                       ‘ enable servo drive
GEARRATIO = .2                ‘ 5:1 reduction
GEAREXT                       ‘ master source Encoder 2 input
GEARON                        ‘ enable gearing
DO                            ‘ continuous loop
LOOP UNTIL 1=2
END
```

8.12 – Manual Control Motion Application

This application requires the motor to follow a manually operated Encoder signal. The Encoder is enabled when input 5 is activated and becomes disabled when input 5 is deactivated. If input 5 is active and input 6 is activated a finer control of the motor is enabled. In the user program Configuration Encoder Folder enter the input encoder line count into the Encoder 2 Line Count text box.

Program Code (manmot.prj included):

```
ACCEL=50                        ‘ input rate limit
ABSPOS=0                        ‘ set Abspos=0 and Encpos2=0
FOLERR =2                       ‘ servo position error limit = 2 units
WNDGS=1                         ‘ enable servo drive
GEARRATIO = 1                   ' 1 revolution of encoder moves 1 revolution of motor
GEAREXT                         ' master source Encoder 2 input
ON IN(5)=1 INTR1                ' go to INTR1 if IN(5)=1 and INTR1 is ON
ON IN(5)=0 INTR2                ' go to INTR2 if IN(5)=0 and INTR2 is ON
INTRON1                         ' enable INTR1
DO                              ' continuous loop
    DO                          ' move the motor with the encoder
        IF IN(6)=1 THEN
            GEARRATIO=.05   ' 20 revolution of encoder moves 1 revolution of motor
        ELSE            ‘ IN(6)=0
            IF GEARRATIO <> 1 THEN
                GEARRATIO=1 ' 1 revolution of encoder moves 1 revolution of motor
            END IF
        END IF
    LOOP UNTIL IN(5)=0
    ‘ ……                        ' program statements
    INTROFF1                    ' optional disable INTR1
    ‘ …...                       ' program statements
    INTRON1                     ' optional enable INTR1
LOOP UNTIL 1=2
END

INTR1:
        GEARON                  ' enable gearing
        INTRON2                 ' enable INTR2
RETURN
INTR2:
        GEAROFF                 ' enable gearing
        INTROFF2                ' disable INTR2
RETURN
```

Section 9

Troubleshooting Guide

Q. I can=t establish communication with my control. What could be the problem?

A. Insure that the connections are correct. See Figure 3.13 for assistance. Check to make sure that the communication parameters are set correctly.

Q. When I try to run my program my motor runs away. What could be wrong?

A. The encoder of the motor may not be functioning or incorrectly connected with the encoder port. Verify the encoder connection, refer to Figure 3.8, and that the encoder is working. You can verify that the encoder is working by either using an oscilloscope or making sure you are receiving an encoder position (encpos).

Q. The OVER TEMPERATURE LED is illuminated. What could be the problem?

A. The over temperature LED tells you that the maximum internal temperature of the drive has been exceeded and the unit has shut down to protect itself. Turn off the control / drive and let it cool then turn it back on. It should work. The other option is to find a way to cool the drive. Frequent over temperature shutdowns may indicate that a cooling problem exists in the control cabinet where the TDC is mounted.

Q. When I press a button connected to my run input the motor does not turn, why?

A. The switch may not be connected properly. See Figure 3.9 for a connection diagram. The motor could be connected improperly. Check the encoder and the phasing. Check to see if there is power to the unit. The program could also be waiting for another input from another device or switch. Hall effect or commutator is wired incorrectly. An over current or temperature condition may have occurred. Check the LED=s on the unit.

Q. After checking my connections and verifying that they are correct, my motor still does not turn when I start the program.

A1. Make sure that the system is tuned. This can be done by checking if there are values Kp, Kd, Ki, Kvff in the configuration screen or check them in terminal mode by querying the controller. If the system is tuned, use a voltmeter and check to see if there is a servo command signal at the Servo CMD output during a move command.

A2. The ENABLE input to the drive has not been properly connected or activated. Insure that the ENABLE input is connected to an active current sinking circuit or switch that is connected with common. **The WNDGS command must be set equal to 1 to enable the drive.**

Q. When my system activates a sensor the controller does not seem to recognize it, why?

A. Check to make sure that you are operating in the correct mode, i.e. if your using an NPN sensor, make sure you are in sink mode, a PNP source mode. Check the connections and make sure you have power to the sensor.

Q. My outputs don=t turn on when they are suppose to turn on. What is the problem?

A. Like the inputs the outputs must also be sink or source. Be sure that the setting for the I/O type is the correct setting for your I/O. NPN, sink; PNP, source.

Q. The OVER CURRENT LED is illuminated. What do I do?

A. If the over current LED is lit, then chances are that there is a short in the cable or the motor. Ohm out the cable for any shorts. If no shorts are found in the cable check the motor for shorts, phase to phase and phase to case.

If more information is needed or additional assistance is required contact Warner Electric - Motors and Controls Division - Bristol Plant Motion Application Engineering Department at 1-800-SUPELEC (1-800-787-3532) between 8:00

a.m. and 5:00 p.m. EST.

Section 10

Glossary

ABSOLUTE MODE - Motion mode in which all motor movements are specified in reference to an electrical home position.

ABSOLUTE POSITION - A data register in the Controller which keeps track of the commanded motor position. When the value in this register is zero, the position is designated "Electrical Home".

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

ACCURACY (of step motor) - The noncumulative incremental error which represents step to step error in one full motor revolution.

AMBIENT TEMPERATURE - The temperature of the air surrounding the motor or drive.

AMPLIFIER - Converts or amplifies low level signals to high voltages and current for use with the motor.

ASCII - (American Standard Code for Information Interchange). A format to represent alphanumeric and control characters as seven-or eight-bit codes for data communications. A table of the ASCII codes appears on page 196.

ATTENTION CHARACTER - <nn, where "nn" is a unique integer from 1-99 (set by use of the unit ID# select switches) that is assigned to a Motion Controller arrayed in a multi-Controller system. The Attention Character directs the program command to the specified Motion Controller.

BACK EMF - The voltage that a permanent magnet generates when it is rotated. This has a linear relationship with speed and is related to the voltage constant or back EMF constant of the motor, K_E which is expressed in units of :

$$\frac{VOLTS}{1000\,RPM}$$

BANDWIDTH - A given range of frequencies that a motion system can respond to commands.

BAUD RATE - The rate of serial data communications expressed in binary bits per second.

BCD - (Binary Coded Decimal), a format to represent the digits 0 through 9 as four digital signals. Systems using thumb wheel switches may program commands using BCD digits. A BCD digit uses a standard format to represent the digits 0 through 9 as four digital signals.

The following table lists the BCD and complementary BCD representation for those digits. The Motion Controller uses the complementary BCD codes because the signals are active low.

BCD code table (0 = low state, 1=high state)

		Complementary
Digit	BCD Code	BCD Code
0	0000	1111
1	0001	1110
2	0010	1101
3	0011	1100
4	0100	1011
5	0101	1010
6	0110	1001
7	0111	1000
8	1000	0111
9	1001	0110

To represent numbers greater than 9, cascade the BCD states for each digit. For example, the decimal number 79 is BCD 0111:1001.

BRAKING TORQUE - The torque that is required to bring the motor from a running condition to a stop. This also describes the torque that is developed during a dynamic braking cycle.

CLEAR - Input or Command to immediately halt all motor motion and program execution.

COLLECTORS (OPEN) - A transistor output that takes the signal to a low voltage level with no pull-up device; resistive pull-ups are added to provide the high voltage level.

COMMUTATION - The function of directing current or voltage to the correct motor phase to produce torque in a motor.

COMMUTATOR - A mechanical device either optical or electromechanical in nature that connect
and / or switch the motor phases to the power source

CYCLE START - Command to initiate program execution.

CYCLE STOP - Command to stop program execution.

DAISY-CHAIN - A method to interface multiple Motion Controllers via RS485 to a single host using only one serial port.

DAMPING - A method of applying additional friction or load to the motor in order to alleviate resonance and ringout. With servo motors, damping can also be adjusted by adjusting the PID loop gains.

DECELERATION - The rate in which the motor speed is decreased from its present speed to a lower speed (specified in units/second/second).

DEVICE ADDRESS - A unique number used to assign which Motion Controller in a multi-drive stepper system is to respond to commands sent by a host computer or terminal. Device addresses from 1 - 99 are set by means of the ID # select switch. "00" is reserved to address all Motion Controllers in a system. Factory default is 01.

DUTY CYCLE - The amount of Aon@ time versus the Aoff@ time. This is usually expressed in terms of a percentage of the Aon@ time versus the total time. This is given by the following equation:

DWELL - See "WAIT".

$$D_{CYCLE} = \frac{T_{ON}}{T_{OFF} + T_{ON}} \text{ x } 100$$

ELECTRICAL HOME - The location where the motor position counter (ABSPOS) is zero.

ENCODER - a mechanical device attached to the motor that provides a pulse output. This output can be used to determine position, speed or acceleration. The encoder may also be an absolute encoder or incremental.

FEEDBACK - A signal that is transferred from the output, in this case the servo 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.

FILTER TIME CONSTANT - The time it takes for a step input to reach 63% of its value at the filter output.

FRICTION - Force that is opposite to the direction of motion as one body moves over another.

FULL-STEP - Position resolution in which 200 pulses corresponds to one motor revolution in a 200 step per revolution (1.8 degree) motor.

HALF-STEP - Position resolution in which 400 pulses corresponds to one motor revolution for a 200 step per revolution (1.8 degree) motor.

HALL SENSOR - A device which is used as feedback to correctly commutate the motor. Typically constructed of a magnetic wheel and hall effect sensors.

HANDSHAKE - A computer communications technique in which one computer's program links up with another's. The Motion Controller uses a software "Xon, Xoff" handshake method. See "XON" below.

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

INCREMENTAL MODE - Motion mode in which all motor movements are specified in reference to the present motor position.

INDEXER - A Microprocessor-based programmable motion controller that controls move distance and speeds; possesses intelligent interfacing and input/output capabilities.

INDEX FROM RUN - See MARK REGISTRATION

INERTIA - Measurement of a property of matter that a body resists a change in speed (must be overcome during acceleration).

INERTIAL LOAD - A "flywheel" type load affixed to the shaft of a step motor. All rotary loads (such as gears or pulleys) have inertia. Sometimes used as a damper to eliminate resonance.

JOG MOVE - moves the motor continuously in a specified direction.

LOAD - This term is used several ways in this and other manuals.

> **LOAD (ELECTRICAL)**: The current in Amperes passing through a motor's windings.
> **LOAD (MECHANICAL)**: The mass to which motor torque is being applied (the load being moved by the system).
>
> **LOAD (PROGRAMMING)**: Transmits a program from one commuter to another. "DOWNLOAD" refers to transmitting a program from a host computer (where a program has been written) to the Motion Controller where it will be used. "UPLOAD" refers to transmitting a program from a Motion Controller back to the host computer.

MARK REGISTRATION - A motion process (usually used in web handling applications) whereby a mark placed on the material is sensed (e.g., through the use of an optical sensor) and, following detection of this mark, the material is moved (indexed) a fixed length.

MECHANICAL HOME - The position where a switch input is used as a reference to establish electrical home.

MOVE TO MECHANICAL HOME - Function which allows the Motion Controller to move the motor and seek a switch to establish electrical home and set Absolute Position = zero.

NESTING - The ability of an active subroutine to call another subroutine. The Motion Controller can nest up to 16 levels.

NON-VOLATILE MEMORY - Data storage device that retains its contents even if power is removed. Examples are EEPROM, flash memory, and battery-backed RAM.

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

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

REGENERATION - A condition when the motor enters a Abraking@ mode. The motor acts as a generator because of the transfer of kinetic energy being converted into electrical energy through the motor.

RESOLUTION - The minimum position command that can be executed. Specified in steps per revolution or some equivalent.

RESOLVER - A feedback device that converts shaft position into an analog signal.

RINGOUT - The transient oscillatory response (prior to settling down) of a step motor about its final position. Note: a small wait or dwell time between moves can alleviate ringout problems.

RMS CURRENT - Root Mean Square Current. In an intermittent duty cycle application, the RMS current is equal to the value of steady state current which would produce the equivalent resistive heating over a long period of time.

RMS TORQUE - Root Mean Square Torque. In an intermittent duty cycle application, the RMS torque is equal to the value of steady state torque which would produce the equivalent resistive heating over a long period of time.

RS232-C - EIA (Electronic Industries Association) communication standard to interface devices employing serial data interchanges. Single-wire connections for transmit and receive, etc.

RS-485 - EIA (Electronic Industries Association) communication standard to interface devices employing serial data interchanges. Two-wire connections (differential circuits) for transmit and receive, etc. Better than RS-232 for long wire runs and multi-drop circuits with many devices.

SINKING - An input that responds to, or output that produces, a "low" level (signal common or low side of the input/output power supply) when active.

SOURCING - An input that responds to, or output that produces, a "high" level (the voltage used for the input/output power supply) when active.

SUBROUTINE - A sequence of lines that may be accessed from anywhere in a program to preclude having to program those lines repetitively. This allows shorter, more powerful, and more efficient programs. See also NESTING.

TORQUE - Product of the magnitude of a force and its force arm (radius) to produce rotational movement. Units of measure are pound-inches, ounce-inches, newton-meters, etc.

TORQUE CONSTANT - A number representing the relationship between motor input current and motor output torque. Typically expressed in units of:

$$\frac{torque}{amp}$$

TRANSLATOR - A motion control device (also called "translator drive") that converts input pulses to motor phase currents to produce motion.

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

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

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

$$\frac{VOLTS}{1000\,RPM}$$

XON / XOFF - A computer software "handshaking" scheme used by a Motion Controller.

The Motion Controller sends an XOFF character (ASCII Code 19) when it receives a command string with a Carriage Return and has less than 82 characters remaining in its host serial port buffer. The Controller sends an Xon when available buffer space reaches 100 characters or in response to an ID attention with adequate buffer space remaining. Since it is impossible for the host device to immediately cease transmissions, the next three characters (subject to the total serial buffer capacity of forty characters) received subsequent to the Motion Controller sending the XOFF character will be stored in the Motion Controller's serial buffer (a memory dedicated to store characters that are in the process of transmission).

Similarly, the Motion Controller will not transmit data if the host device has sent an XOFF character to the Controller; Motion Controller transmissions will resume when the Controller receives an XON character.

ASCII Table

ASCII Char	Dec Code	ASCII Char	Dec Code	ASCII Char	Dec Code	ASCII Char	Dec Code
Null	0	Space	32	@	64	`	96
SOH	1	!	33	A	65	a	97
STX	2	A	34	B	66	b	98
ETX	3	#	35	C	67	c	99
EOT	4	$	36	D	68	d	100
ENQ	5	%	37	E	69	e	101
ACK	6	&	38	F	70	f	102
BELL	7	›	39	G	71	g	103
BS	8	(	40	H	72	h	104
HT	9	)	41	I	73	I	105
LF	10	*	42	J	74	j	106
VT	11	+	43	K	75	k	107
FF	12	,	44	L	76	l	108
CR	13	-	45	M	77	m	109
SO	14	.	46	N	78	n	110
SI	15	/	47	O	79	o	111
DLE	16	0	48	P	80	p	112
DC1	17	1	49	Q	81	q	113
DC2	18	2	50	R	82	r	114
DC3	19	3	51	S	83	s	115
DC4	20	4	52	T	84	t	116
NAK	21	5	53	U	85	u	117
SYNC	22	6	54	V	86	v	118
ETB	23	7	55	W	87	w	119
CAN	24	8	56	X	88	x	120
EM	25	9	57	Y	89	y	121
SUB	26	:	58	Z	90	z	122
ESC	27	;	59	[	91	{	123
FS	28	<	60	\	92	|	124
GS	29	=	61	]	93	}	125
RS	30	>	62	^	94	~	126
DEL	31	?	63	_	95	DEL	127

Appendix A
CE Compliance
Installation Requirements and Information

Certain practices must be followed when installing a TD servo drive or a TDC servo drive/controller in order to meet the CE Electromagnetic Compatibility (EMC) Directive (89/336/EEC) and the Low Voltage Directive (73/23/EEC). The TD family of products are components intended for installation within other electrical systems or machines. The system or machine builder must ensure their system or end product complies with all applicable standards required for that equipment, including overall CE certification. Following these practices will help ensure (but cannot guarantee) that the machine in which these components are utilized will meet overall CE requirements.

Electromagnetic Compatibility Directive

In order to meet the various EMC Standards, all wiring must be done in accordance with the practices shown in Figure 1.

With the addition of a suitable ac line filter, such as Corcom part number 15ET1 (connected externally), the TD drive and TDC drive/controller meet all the applicable EMC emission and immunity standards on a Astand-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 TD or TDC must be placed inside a metal enclosure, as shown in Figure 1.

Low Voltage Directive

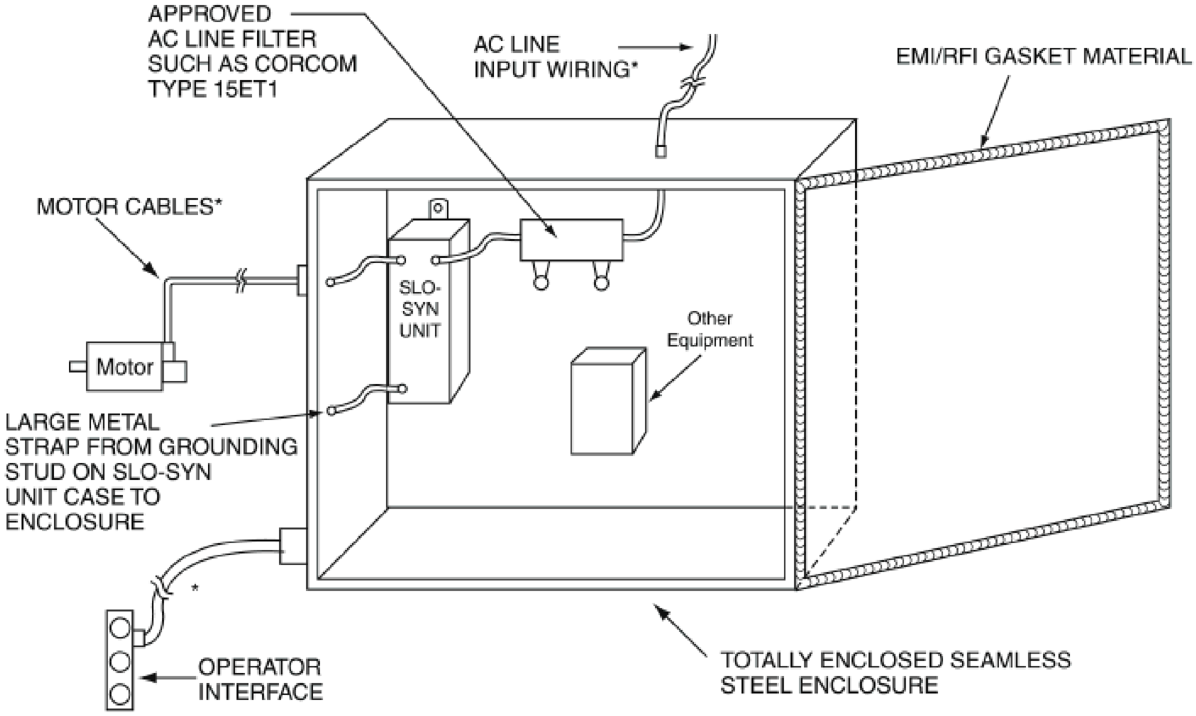
1) These drives are to be operated in a pollution degree 2 environment as described in standard EN50178.

2) All of the control inputs and outputs are isolated from the main input power with a Abasic insulation rating@; e.g., their impulse withstand voltage capability is 2.5kV (1.2 / 50 us) 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 the machine-builder=s responsibility to provide this protection, if needed.

3) For electrical safety, and to protect personnel against direct contact with live electrical parts, the terminal cover (provided with the unit) MUST be installed over the AC input, motor output, and External REGEN terminals.

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

Figure 1:
Installation for EC EMI/RFI Compliance



* ALL WIRING RUNS THROUGH METAL JACKETED CONDUIT WHICH ARE ATTACHED TO ENCLOSURE WITH CLAMPS MAKING SOUND ELECTRICAL CONTACT.

NOTES:

1) All metal mating surfaces within the enclosure, and any mounting plates should be cleaned of paint, anodizing and coating material for proper electrical bonding. This includes mounting of SLO-SYN unit, line filter, and any other equipment.

2) If mounting plates are used, proper electrical contact to the main enclosure must be maintained. Using copper straps with length-to-width ratios less than 3 is optimum.

WARRANTY AND LIMITATION OF LIABILITY

Superior Electric (the "Company"), Bristol, Connecticut, warrants to the first end user purchaser (the "purchaser") of equipment manufactured by the Company that such equipment, if new, unused and in original unopened cartons at the time of purchase, will be free from defects in material and workmanship under normal use and service for a period of one year from date of shipment from the Company's factory or a warehouse of the Company in the event that the equipment is purchased from the Company or for a period of one year from the date of shipment from the business establishment of an authorized distributor of the Company in the event that the equipment is purchased from an authorized distributor.

THE COMPANY'S OBLIGATION UNDER THIS WARRANTY SHALL BE STRICTLY AND EXCLUSIVELY LIMITED TO REPAIRING OR REPLACING, AT THE FACTORY OR A SERVICE CENTER OF THE COMPANY, ANY SUCH EQUIPMENT 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. THE COMPANY RESERVES THE RIGHT TO SATISFY SUCH OBLIGATION IN FULL BE REFUNDING THE FULL PURCHASE PRICE OF ANY SUCH DEFECTIVE EQUIPMENT. This warranty does not apply to any equipment which has been tampered with or altered in any way, which has been improperly installed or which has been subject to misuse, neglect or accident.

THE FOREGOING WARRANTY IS IN LIEU OF ANY OTHER WARRANTIES, EXPRESS OR IMPLIED, INCLUDING, WITHOUT LIMITATION, ANY IMPLIED WARRANTY OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE, and of any other obligations or liabilities on the part of the Company; and no person is authorized to assume for the Company any other liability with respect to equipment manufactured by the Company. The Company shall have no liability with respect to equipment not of its manufacture. **THE COMPANY SHALL HAVE NO LIABILITY WHATSOEVER IN ANY EVENT FOR PAYMENT OF ANY INCIDENTAL OR CONSEQUENTIAL DAMAGES, INCLUDING, WITHOUT LIMITATION, DAMAGES FOR INJURY TO ANY PERSON OR PROPERTY.**

Written authorization to return any equipment or parts thereof must be obtained from the Company. The Company shall not be responsible for any transportation charges.

IF FOR ANY REASON ANY OF THE FOREGOING PROVISIONS SHALL BE INEFFECTIVE, THE COMPANY'S LIABILITY FOR DAMAGES ARISING OUT OF ITS MANUFACTURE OR SALE OF EQUIPMENT, OR USE THEREOF, WHETHER SUCH LIABILITY IS BASED ON WARRANTY, CONTRACT, NEGLIGENCE, STRICT LIABILITY IN TORT OR OTHERWISE, SHALL NOT IN ANY EVENT EXCEED THE FULL PURCHASE PRICE OF SUCH EQUIPMENT.

Any action against the Company based upon any liability or obligation arising hereunder or under any law applicable to the sale of equipment, or the use thereof, must be commenced within one year after the cause of such action arises.

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

Distribution Coast-To-Coast and International

Superior Electric SLO-SYN products are available worldwide through an extensive authorized distributor network. These distributors offer literature, technical assistance and a wide range of models off the shelf for fastest possible delivery and service.

In addition, Superior Electric sales and application engineers are conveniently located to provide prompt attention to customers' needs. Call Superior Electric customer service for ordering and application information or for the address of the closest authorized distributor for Superior Electric's SLO-SYN products.

In U.S.A. and Canada
Superior Electric

- **Customer Service: 1-800-787-3532**
- **Product Application: 1-800-787-3532**
- **Product Literature Request: 1-800-787-3532**
- **Fax: 1-800-766-6366**
- **Web Site: www.warnernet.com**

383 Middle Street
Bristol, CT 06010
Tel: 860-585-4500
Fax: 860-589-2136

In Europe
Warner Electric (Int.) Inc.

- **Tel: 41 021 631 33 55**
- **Fax: 41 021 636 07 04**

La Pierreire
CH-1029 Villars-Ste-Croix, Switzerland

Superior Electric
383 MIDDLE STREET BRISTOL, CT 06010
(860) 585-4500 FAX: (860) 589-2136

REGISTERED COMPANY
ISO 9001
SGS