Whedco IMC-313P-X-D

DspMotion Independent Motion Controller

$3500.00

In Stock

Qty Available: 1

Used and in Excellent Condition

Open Web Page

https://www.artisantg.com/72230-1

Whedco

DspMotion™

INDEPENDENT MOTION CONTROLLERS
USER'S MANUAL (174/7)

MODELS

IMC-105X-1-C IMC-200X-X-C IMC-316X-X-C
IMC-105E-1-C IMC-313E-X-C IMC-316E-X-C IMC-31CE-2-C IMC-31PE-2-C

Price $75.00
Pub. No. 174 ■ Revision 7 ■ Copyright 1995

WHEDCO INCORPORATED
1050 Highland Drive ■ Ann Arbor, MI 48108
Technical Support (734) 665-7540 ■ Facsimile (734) 665-6694

Since this equipment can be applied in many diverse situations and in conjunctions with equipment from other vendors, the user and those responsible for specifying this equipment must determine for themselves its suitability for the use intended. In no event, shall Whedco Incorporated be liable for loss of use, profit or consequential damages, or damage to other equipment, resulting from the use of this equipment.

The figures and examples in this manual are designed to demonstrate general concepts for the installation and maintenance of this equipment. The users should always verify interconnection requirements to and for other equipment as well as confirm installation and maintenance requirements for the specific application. In no case, shall Whedco Incorporated be liable for actual use based on the guidelines mentioned herein.

Table of Contents

Product Overview

Features

- All digital design utilizes state-of-the-art digital signal processing (DSP) technology
- Integrated "plug and play" design incorporates control, drive, user I/O and optional operator interface; power supplies included, allowing simple Vac input
- 32-bit CISC microprocessor and fully featured command set supports multitasking, floating point math and conditional program execution
- Stand-alone operation or peripheral to other control devices
- Easy interface with PLCs or computers
- Zero dither, zero drift servo drive design with auto-tuning
- Microstepping drive provides 50,000 steps per revolution for smooth operation and improved resolution, 170 Vdc bus provides maximum high speed performance
- Optional analog input for a variety of uses, such as input from pressure and torque transducers
- Optional auxiliary encoder input for applications requiring encoder tracking such as phase-locked-loop and electronic gearing or for use with a secondary feedback device

Applications

The **DspMotion®** Controllers can be used in a wide variety of common servo and stepping motor applications, some examples of which are listed below:

- high speed, accurate indexing operations
- start-stop, continuous and synchronized conveyors
- flying shears
- precision machining
- flexible fixturing
- package wrapping and random infeed
- closed loop control with analog output transducers
- walking beam

General Description

The Whedco family of **DspMotion®** Controllers provides state-of the-art motion control for many types of machinery requiring the control of one or more axes of servo and/or stepping motors. The **DspMotion®** Controllers provide the all digital, real-time response needed for today's automation applications. In addition, Whedco's self-contained and compact packaging philosophy results in a unit which is cost-effective for machine builders and users alike by minimizing costly point-to-point wiring and panel space. Like all other Whedco stand-alone motion controllers, **DspMotion®** is programmed using an easy-to-understand, mnemonic command set. Once programmed, the unit and associated mechanical system will stand-alone as an "island of automation," supervising and executing all motion control activities.

DspMotion® uses a technology called digital signal processing (DSP), which eliminates the analog elements of the motor control and drive package while improving system update rates. This means no drift, no dither, and high performance machine response. The motion loop update rate is 122 microseconds, meaning motor control is stiff, responsive, and smooth. **DspMotion®** provides a straightforward and no-nonsense approach for machine designers to incorporate leading edge motion control technology into a wide variety of automation machinery. The **DspMotion®** family is highly recommended for any application which requires one of the following motion control capabilities . . .

Accurate Velocity Control

Servo and stepping motors are necessary in many applications requiring accurate velocity control. Unlike conventional variable speed motors, Whedco servo and stepping motor systems provide long-term velocity error of less than a tenth of a percent. Tighter process constraints imposed on many system designers make this a mandatory feature in many velocity control applications. In addition, Whedco servo and stepping motor systems operate at higher speeds than typical d.c. motors. Whedco systems are available to operate at speeds in excess of 5000 r.p.m. Finally, Whedco servo and stepping motor systems operate at a constant speed regardless of input line voltage, a critical feature in maintaining accurate velocity control.

Point-to-Point Positioning

Numerous applications require point-to-point position control where a relative or absolute position destination can be specified. The **DspMotion®** Controllers offer many possibilities for this type of move. Position moves can be preprogrammed and executed in response to a discrete input. Alternatively, destination position can be conditional based on program variables or external commands received from a host computer.

Multiple Speed Position Moves

Certain applications require different motor speeds when specific positions are achieved. This is common in applications such as drilling, where a rapid infeed is followed by a slower drill-to-depth move. The **DspMotion®** Controllers include complete capabilities to define this type of move. Exact behavior of the move can be described such that each segment of the move is completed without blending any of the move segments. Such moves can also include outputs which turn on as specific speeds and/or positions are achieved for auxiliary operations such as turning on coolant.

High-Speed Position Capture

The performance of many applications, in particular those requiring conditional indexing, can be improved by obtaining real-time position information. This is often difficult given inherent processing latencies in computer systems. The **DspMotion®** Controllers contain a dedicated high-speed input for a position sensor. This input will capture and store the position of the motor at the time this input is received in less than 500 nanoseconds.

Torque-Limited Moves

In servo systems, not only can the speed and position of the motor be controlled, but also the output torque. This is useful in applications where excess force can damage parts or where the assembly specification includes a force with which the part must be inserted. The **DspMotion®** Controllers allow the user to control directly the torque parameter in conjunction with position. This capability is not available in stepping motor systems.

Additional Capabilities of the *Enhanced* Option . . .

Whedco's *enhanced* version incorporates both an analog input and an auxiliary encoder input along with an enhanced command set for controlling motions which are synchronized with these inputs. With the addition of these inputs and an expanded command set, the following additional capabilities are provided:

Electronic Gearing

Many applications require the synchronization of one axis with an external axis, that is, an axis that will not be controlled by the Motion Control system, so that the controlled motor runs at a ratio which is proportional to the external axis. This is typical in applications such as conveyors. In such applications, an incremental encoder is mounted to the external axis. An axis synchronized to this external incremental encoder input behaves in a manner which is similar to mechanical gearing, hence the term electronic gearing.

Phase-Locked-Loop

Certain applications must coordinate the phase of one axis with the phase of another. Such an application would, like the electronic gearing case, incorporate an external axis which is monitored by the Motion Controller via an incremental encoder, but it would also include added feedback about the location via a device such as a photocell. The photocell requires a high speed input know as "position capture." The controller provides inputs for both the incremental encoder from the external axis and the high speed input for position capture, and in this manner can synchronize the phase of one axis with another.

Secondary Position Feedback

The auxiliary incremental encoder input can be used for secondary position feedback. This is useful in applications requiring very accurate positioning. It can be used to eliminate position uncertainties between the motor and the load caused by such things as backlash.

Closed Loop Pressure and Torque Control

The analog input can be used to read input from pressure and torque transducers to provide closed loop pressure and torque control. In this type of application, the servo motor can be commanded to exert a certain amount of force (pressure or torque). The amount of force actually exerted can then be modified based on the reading from the transducer. Torque or pressure limited moves can be used in conjunction with position moves for more complex applications.

System Configuration

Key design features have been incorporated into the **DspMotion®** family to provide the flexibility, performance, and serviceability demanded by the industrial user. These features include:

- common capabilities and programming language for servo and stepping motors
- complete software configuration of critical drive parameters such as peak and continuous current
- optional plug and play operator interface
- industrially hardened discrete input and outputs rated up to 24 Vdc
- quick disconnect terminals for ease of installation and replacement
- elimination of switch settings wherever possible
- common mechanical specifications among all units

Peripheral to Programmable Logic Controller via Discrete I/O

The **DspMotion®** Controllers include dedicated inputs and outputs ideally suited to execute predefined motions in response to a discrete input. The controller is equipped with edge-sensitive inputs which, if programmed to operate accordingly, cause the controller to execute the associated set of commands when the input is activated. The unit can be programmed to turn on an output when the execution of the sequence is complete and the controller is ready for the next sequence. Up to 1000 such sequences can be stored in the controller for subsequent execution.

Peripheral to a Host Controller via Serial Communication

The **DspMotion®** Controllers do not require that their programs be compiled prior to execution. This means that commands, parameters, and status information can be sent back and forth interactively between the motion controller and a host controller. This configuration is ideal for in-process and test applications in which motion parameters can change. The **DspMotion®** Controllers come standard with a serial port compatible with RS-232 and RS-422. Multiple units can be addressed over this serial-data link. This mode of operation is not precluded by use of the optional operator interface. The link is fault tolerant to controller node failure and will continue operation if power to a particular unit is lost.

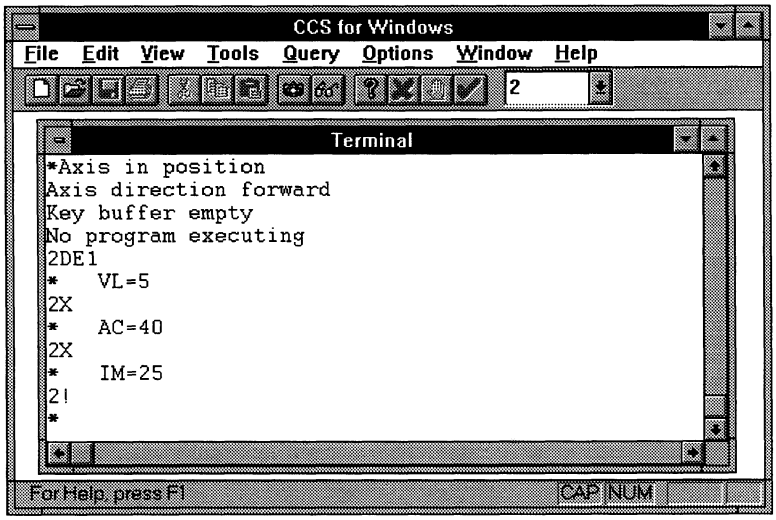
Stand Alone System

The **DspMotion®** Controllers include a comprehensive command set capable of conditional logic, fault handling, and power-up routines. It is possible for the **DspMotion®** Controllers to provide complete stand-alone machine control in applications requiring a limited amount of discrete inputs and outputs.

Programming

The **DspMotion®** Command Set provides a comprehensive set of programming tools for advanced motion control applications and stand-alone machine control. The programming environment is multitasking and can support up to four programs. In addition, up to 100 motion blocks are available as subroutines to be called for specific motion tasks. The command set also provides tools for easy customization of the optional operator interface. Most importantly, it is not necessary to compile these programs. On-line programming capability is provided through the resident line editor. This on-line capability provides a significant time savings in machine set-up by allowing an interactive debugging environment.

For off-line program development and file management, Whedco offers optional Windows compatible software development tools.

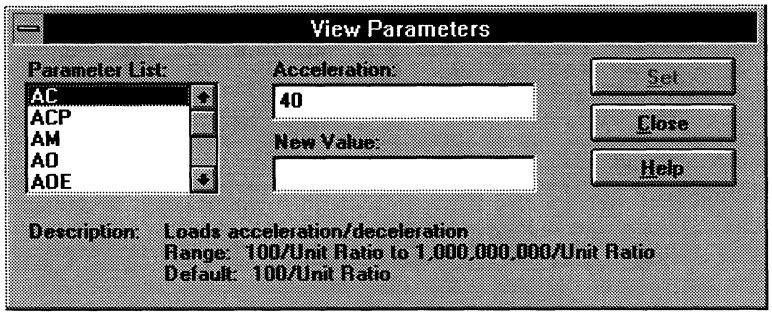


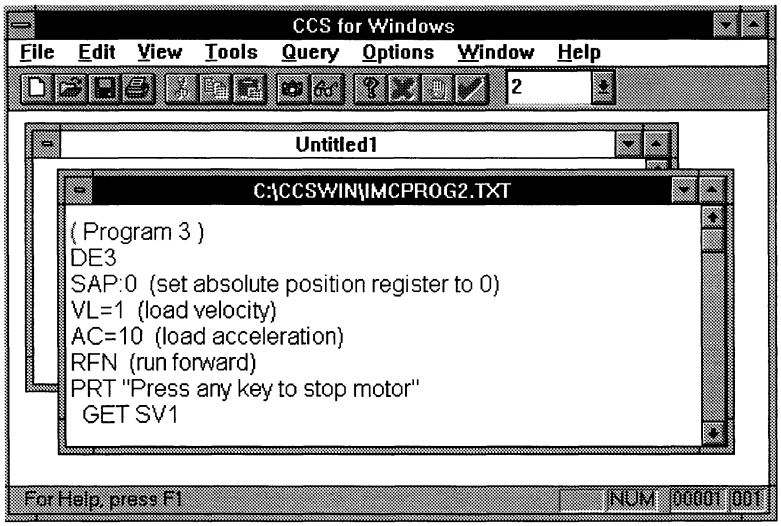
Basic Terminal

This software allows easy communication with the controller from a PC/AT compatible computer providing a terminal emulator for direct communication with any controller in the serial-data link chain.

Typical Parameter

On-line help provides complete explanation of user parameter settings and complete descriptions of program commands.



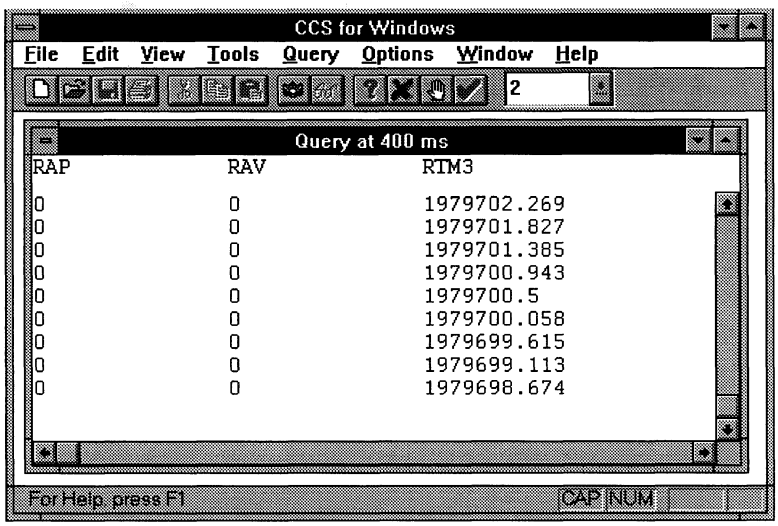


Text Editor

A resident text editor is included for off-line program development along with upload-down utilities for program management.

Query Mode

A query function provides on-screen, real-time reporting on machine parameters.



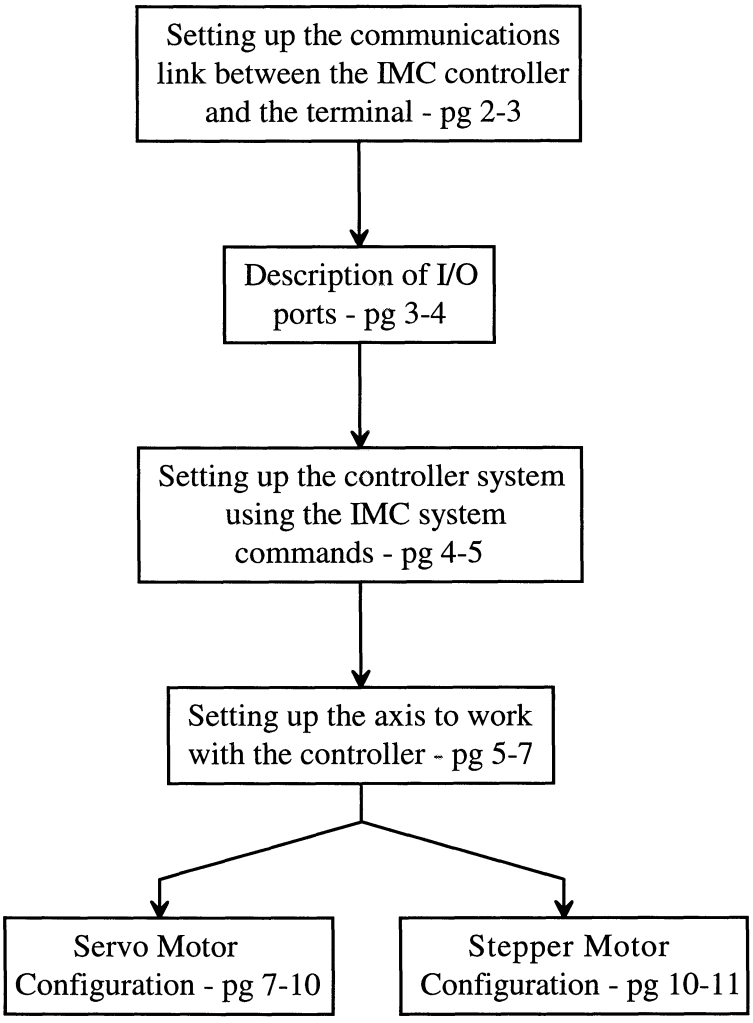
A program consists of simple mnemonics to represent motion and machine control parameters. On the following page is an example representative of a typical program. When executed, this program will wait for one of the digital inputs 1, 2 or 3 to be active. Once one of them is active, the program will jump to the respective subroutine at label 10, 20 or 30. The subroutine at label 10 executes motion block 1 which runs the motor in the forward direction. The subroutine at label 20 runs the motor in the forward direction, and once digital input 2 is active again, it stops the motor. The subroutine at label 30 homes the motor to the marker and sets that position equal to absolute position 0. This subroutine will be bypassed if the motor is not in position, i.e. stopped. The term "units" refers to engineering units.

Typical Program Example

Main		DE1	(define program 1)
Program	1	IF EG1 GS10	(conditionally gosub 10 if positive edge sensitive)
			(digital input 1 becomes active)
		IF EG2 GS20	(conditionally gosub 20 if positive edge sensitive)
			(digital input 2 becomes active)
		IF EG3 GS30	(conditionally gosub 30 if positive edge sensitive)
			(digital input 3 becomes active)
Subroutines		GT1	(unconditionally goto 1)
	10	MEX1	(execute motion block 1)
		RET	(return from gosub)
	20	VL=10	(load velocity of 10 units/sec)
		AC=40	(load acceleration of 40 units/sec^2)
		RFN	(run motor forward)
		WT EG2	(wait for positive edge sensitive digital input 2 to)
			(become active)
		ST	(stop all motion)
		RET	(return from gosub)
	30	IF N IP GT35	(conditionally goto 35 if motor is not in position)
		HFM	(home forward to marker)
		WT IP	(wait for motor to be in position)
		SAP:0	(set absolute position register to 0)
	35	RET	(return from gosub)
Program End		ED	(end program 1 and exit program editor)
Motion		MB1	(define motion block 1)
Block		AM=50	(load absolute move position of 50 units)
		VL=20	(load velocity of 20 units/sec)
		AC=40	(load acceleration of 40 units/sec^2)
		RAN	(run motor to absolute move position)
		AM=100	(load absolute move position of 100 units)
		VL=5	(load velocity of 5 units/sec)
		RAN	(run motor to absolute move position)
		MED	(end motion block 1 and exit program editor)
		WT IP	(wait for motor to be in position)
		MED	(end motion block 1 and exit motion block editor)

Product Manual

1. IMC Controller Setup



1. IMC Controller Setup

A. Hardware

1. Communications

This section describes how to set up the communications link between the IMC controller and the terminal. Three main things must be done in order to ensure proper communication between the controller and the terminal:

a. the address of the unit must be set;
b. the serial baud rate must be set; and
c. the data link connectors must be properly wired and attached to the controller(s).

a. Unit Address

The address of the IMC unit is set by the DIP switch. Using switch positions 1 through 5, the address of the unit can be set from 0 through 9, or A through V. This gives 32 possible addresses to work with. The table below shows which DIP switch setting is to be used for a specific address.

Unit Address	1	2	3	4	5
0	R	R	R	R	R
1	L	R	R	R	R
2	R	L	R	R	R
3	L	L	R	R	R
4	R	R	L	R	R
5	L	R	L	R	R
6	R	L	L	R	R
7	L	L	L	R	R
8	R	R	R	L	R
9	L	R	R	L	R
A	R	L	R	L	R
B	L	L	R	L	R
C	R	R	L	L	R
D	L	R	L	L	R
E	R	L	L	L	R
F	L	L	L	L	R

Unit Address	1	2	3	4	5
G	R	R	R	R	L
H	L	R	R	R	L
I	R	L	R	R	L
J	L	L	R	R	L
K	R	R	L	R	L
L	L	R	L	R	L
M	R	L	L	R	L
N	L	L	L	R	L
O	R	R	R	L	L
P	L	R	R	L	L
Q	R	L	R	L	L
R	L	L	R	L	L
S	R	R	L	L	L
T	L	R	L	L	L
U	R	L	L	L	L
V	L	L	L	L	L

b. Serial Baud Rate

The serial baud rate is also set by the DIP switch. Using switch positions 6 and 7, you can set the baud rate to 1200, 9600, 19200, or 38400. Note that the baud rate can also be set by the BD command after the controller is powered up. The DIP switch setting therefore gives the default value for BD. The table that follows shows which DIP switch setting is to be used for what baud rate.

Baud Rate	6	7
1200	R	R
9600	L	R
19200	R	L
38400	L	L

When using the display, you must also set the baud rate on the DIP switch of the display. Using switch positions 1 and 2, you can set the baud rate to 1200, 9600, 19200, or 38400. The table that follows shows which DIP switch setting is to be used for what baud rate.

Baud Rate	1	2
1200	U	U
9600	D	U
19200	U	D
38400	D	D

c. Serial data link

All IMC units are compatible with RS-232 and RS-422 serial communication standards. In order to properly wire the connectors, refer to the diagram entitled "Serial Data Link" in Appendix D for your specific IMC controller. Once the connectors are wired correctly, the cable from the terminal should be attached to the "Host" port on the first IMC. Then, if you wish to daisy chain other IMC's, the cable from the "Link" port should be attached to the "Host" port of the next IMC, and so on. The last IMC should have no cable attached to the "Link" port. Note that when using RS-232 protocol, you can have up to 32 units without the display or 31 units with the display. When using RS-422 protocol, you can have only 31 units whether using the display or not.

2. Analog & Digital I/O

There are 24 ports on the front of the controller which are used as a means of interface with other machine control devices. Ports 1 through 12 are general purpose digital inputs/outputs. The first 6 are strictly inputs while the next 6 are combined inputs and outputs. These are set and reset by the RSD, STD and DO commands. The inputs can also be filtered using the DIF and DFT commands, in order to deal with problems such as contact bounce. Digital outputs 11 and 12 are also used as set point outputs, with output 11 being set point A and output 12 being set point B. These can be defined, set and reset with the set point commands, which are SPA*n*B, SPA*n*E, SPA*n*F, SPB*n*B, SPB*n*E, and SPB*n*F. See the command summary for more information.

The next 5 ports, which are 13 through 17, and also ports 22 through 24, are specific purpose digital inputs/outputs. They are as follows:

- Port 13 is the Home input, which is used to tell the controller when the axis has reached its home position defined by the user.
- Ports 14 and 15 are the Forward Overtravel and Reverse Overtravel inputs, which are used to limit the motion of the axis in the forward or reverse direction.
- Port 16 is the enable input, which is used to enable the controller.
- Port 17 is the OK output, which is active whenever the controller is not faulted.
- Port 22 is the analog output, which can be assigned to different values using the AO command. For more information, see the command summary.
- Ports 23 and 24 are used together as the position capture input. This is a differential input which works from 5 to 24 volts.

The rest of the ports, 18 through 21, are used as reference voltages for the inputs and outputs. See Appendix D for information on how to make the attachments, and refer to Appendix E for specifications on the inputs and outputs.

B. Software

1. System Configuration

This section describes how to set up the controller system once the necessary hardware connections have been made. First, a command summary is given for the system commands, which are used, as the name implies, to set up the system and for other system operations. Then, the following are explained:

a. how to enter and change the password;
b. how to set up the serial port format;
c. how to set up the real time clock;
d. how to allocate variable space; and
e. how to store the memory contents in non-volatile storage.

System Command Summary

mnemonic	*description*
FMW	downloads firmware
RVM	retrieves user memory
SVM	saves user memory
CLM	clears all user memory and resets parameters to factory defaults
SF	secures user memory space
ATE	enables auto retrieving of user memory
FP	assigns floating point variable allocation
BD	assigns baud rate of serial port
HSE	enables XON, XOFF handshake protocol on serial port
CIE	enables computer interface format on serial port
DSE	enables display format on serial port
KY	puts character into key buffer
EKB	empties key buffer
PW	prompts for password
PWC	prompts for password change
RSF	resets faults
STF	sets fault
SRTC	sets real time clock

a. Password (PW, PWC)

Once the controller has been powered up, the password should be entered using the PW command. The password can be up to 10 characters long. If the PW command has not been issued and the correct password not entered, only diagnostic commands can be entered.

The password can be changed with the PWC command. Once you issue the PWC command, the controller will respond with a series of prompts. Once having responded to these prompts, the password will be

changed to the new one you have entered. For more information, see the PWC command in the command summary.

b. Serial Port Format (CIE, HSE, DSE)

In order to make the communications link work after it has been physically connected, it must be known whether the link will use XON, XOFF handshake protocol. If it is, HSE must be set to 1. This enables the XON, XOFF handshake protocol on the serial port. If not, HSE should be set to 0.

Another setting to take into account is CIE, which enables the computer interface format on the serial port. If you set CIE to 1, the controller will report status register and error message by its register value. This value determines the type of message, and in the case of all except for command messages, how many messages are applicable to the specific situation. If CIE is set to 0, all controller messages will be given as an English statement. For a listing and description of the controller messages, see Appendix C, "Message Charts".

Lastly, if you are using a display with your set up, you will probably want to set DSE to 1. This enables the display format on the serial port. This means that all commands that make use of the display (ASC, PRT, PNT, and OUT) will send output to the display. If DSE is set to 0, then all output will be sent to the terminal. You should do this if you do not have a display attached.

c. Real time clock (SRTC, RTC)

Once the controller is first powered up, you should set the real time clock using the SRTC command. The real time clock keeps track of the date and time. It can be printed to the terminal using the RTC command, or can be loaded into a variable using the RTC operand. The real time clock provides a formatted output when set to a string variable, and provides the number of seconds from 1/1/94 when set to an integer variable.

d. Variable allocation (FP)

In order to assign floating point variable allocation, you must use the FP command. Once the memory is cleared using the CLM command, FP is set to 1024, which means that half of the variable space is allocated for floating point variables. The FP command has the effect of writing over part of the space normally allocated for integer variables. The floating point variables will range from 1 to FP, and the integer variables will range from 1 to 4,096 - (2*FP). See the FP command in the command summary for more information.

e. Memory storage (ATE, SVM)

The user memory that contains programs, motion blocks and parameters is battery backed. In case of a battery malfunction, you may take precaution by saving user memory to the non-volatile memory chip installed in the controller. This can be done using the SVM command. Before doing this, however, you may want to set ATE to 1. This will enable auto retrieving of user memory from non-volatile storage upon power up. This ensures that user memory will not be erased inadvertently by a possible battery malfunction.

2. Axis Configuration

This section describes how to set up the axis to work with the controller. First, a command summary is given for the axis commands, and then the following are explained:

a. how to generally configure the axis;
b. how to configure the servo motor; and
c. how to configure the stepper motor.

Axis Command Summary

mnemonic	*description*
PFE	enables position feedback
PWE	enables position register wrap
FO	loads forward software overtravel limit
RO	loads reverse software overtravel limit
MD	assigns direction of motor for forward moves
AUR	loads axis unit ratio
APL	loads axis position length
FB	loads following error bound
PB	loads in-position band
FR	loads axis feedback resolution
AR	loads amplitude of resolver excitation
PR	loads motor poles to resolver poles ratio
CO	loads commutation angle offset
CA	loads commutation angle advance
CC	loads continuous current in percent of maximum
PC	loads peak current in percent of maximum
SC	loads power save current percentage
TC	loads torque limit current in percent of maximum
TLE	enables torque limit
POE	enables power output stage
KP	loads proportional control gain
KI	loads integral control gain
KD	loads derivative control gain
KA	loads acceleration feedforward
FT	loads filter time constant
CT	loads correction time constant
BL	loads backlash constant
DB	loads deadband constant
PF	assigns position feedback input type
MS	automatically sets up motor constants
KS	automatically sets up control constants

a. General Configuration

This section describes:

1. how to set the axis unit ratio and feedback resolution;
2. how to define motor direction and whether the position register is to wrap around; and
3. how to set the following error bound, in-position band and software overtravel limits.

1. Axis Unit Ratio (AUR) and Axis Feedback Resolution (FR)

Many axis and register commands load values of position pulses in order to define axis motion. Most of these can be changed to engineering units, such as revolutions, inches or millimeters by changing the value of AUR (axis unit ratio). For example, in the case of a resolver feedback servo system, the axis unit ratio can be set to 4,096 pulses/unit. This will change the units of the appropriate axis and register commands to revolutions. For more information, see the AUR command in the command summary.

It is necessary to set the axis feedback resolution, FR, so that the controller knows the number of pulses/revolution of the motor being used. For resolver feedback servo systems, the default value for FR is 4096 pulses/revolution, and for encoder feedback servo systems, it is 1000 pulses/revolution. The default value will most likely be the one you use. For stepping systems, you cannot vary the axis feedback resolution; is it fixed at 50,000 pulses/revolution. See the FR command in the command summary for more information.

2. Motor Direction (MD), Position Register Wrap (PWE) and Axis Position Length (APL)

The MD command is used to set the forward direction of the motor. It can be either clockwise (CW) or counterclockwise (CC).

In applications where the axis is to run for an indefinite period of time in one direction, it is possible for the position register to overflow. In order to bypass this inconvenience, position register wrap should be enabled by setting PWE to 1. Then, a position length can be set by using the APL command. This length will be the value that the position register reaches before wrapping around. This is especially useful in applications where a rotary table is used. See APL in the command summary for more information.

3. Following Error Bound (FB), In-Position Band (PB) and Software Overtravel Limits (FO, RO)

The following error bound is set by the FB command. In some cases, a motion or a combination of motions can lead to some following error in the axis. It is therefore important to set a bound on how much following error can be allowed before the axis faults due to excessive following error. The following error bound, FB, is equal to the number of units of following error that will be allowed before the controller faults due to excessive following error.

The in-position band is set by the PB command. This defines a bound on how many units the axis can be off of the command position, CP, to be considered in position.

In addition to the hardware overtravel inputs, there are software overtravel limits. The forward overtravel and reverse overtravel limits, FO and RO, are used to set a bound on how far the axis will travel in one direction or another. When moving the axis with a home command (HFH, HFM, HFO, HRH, HRM, or HRO), the software overtravels are not monitored.

b. Servo Motor Configuration

This section describes:

1. how to set the current settings;
2. how to define torque limit;
3. how to set up the motor constants; and
4. how to set up the control constants.

1. Current Settings (CC, PC)

There are several different current settings that should be made to setup a servo system. CC and PC should be set, and TC should be set if you will be doing torque limited moves (see "Torque Limit"). The continuous current in percent of the continuous current rating of the drive is set by CC, and the peak current in percent of the peak current rating of the drive is set by PC. You will need to set CC to limit the continuous current of the drive to the continuous rating of the motor. For example, if the continuous rating of the motor is 3 amps, and if the continuous rating of the drive is 6 amps, CC should be set to 50.

2. Torque Limit (TLE, TC)

In servo systems, not only can the speed and position of the motor be controlled, but also the output torque. This is useful in applications where excess force can damage parts or where the assembly specification includes a force with which the part must be inserted. In order to accomplish torque limited moves, the torque

limit must be enabled by setting TLE to 1. Also, the torque limit current should be set. TC is the torque limit current in percentage of the continuous rating of the drive.

3. Motor Constants (MS/CO, PR; AR, CA)

When using a brushless servo motor, you must set up the motor constants in order to make the servo motor that you are using able to work with the IMC unit. This can be done automatically by using the MS command. You should make sure that the motor is not connected to the load when using this command. Once the MS command is executed, the controller will calculate the following constants for you:

- CO - commutation angle offset
- PR - motor poles to resolver poles ratio.

Alternatively, the motor constants can be set up manually by simply using the commands CO and PR to load them in.

In addition, there are some constants that cannot be automatically set up by the controller, namely the amplitude of resolver excitation, AR, and the commutation angle advance, CA. The amplitude of resolver excitation is determined by the transformation ratio of the resolver. The default value for AR, which is 1, will work with most Whedco servo motors. The commutation angle advance compensates for the lag in the commutation angle at high speed introduced by the inductance of the motor. Most systems will not need to make this compensation; therefore, CA can be left at the default value of 0 in most systems. Again, AR and CA are not calculated by the MS command. For more information about these commands, see the command summary.

4. Control Constants (KS/KP, KI, KD, KA, FT)

In all servo systems, it is also important to set up the control constants. This can be done automatically with the KS command. You should make sure that the motor has a load attached before issuing the KS command. Once this command is executed, the controller will calculate the control constants for you. These control constants are:

- KP - proportional control gain
- KI - integral control gain
- KD - derivative control gain
- KA - acceleration feedforward constant
- FT - filter time constant

If you are not able to automatically tune using the KS command, you will need to select the control constants that will ensure the best system performance. This can be accomplished by either (i) calculating the control constants using the axis feedback resolution, FR, and the torque to inertia ratio of your motor or (ii) using an oscilloscope to monitor the motor performance and changing the control constants appropriately until the desired motor response is obtained. The following section shows how to tune your servo motor using method (ii). This can be more advantageous than method (i) because you can accurately monitor the system. Although it may take more time, manually tuning the motor will give you more control over the performance of the motor. You will need to use an oscilloscope with an input attached to the analog output (port 22) of the IMC.

After turning on and setting up the scope, you should enable the analog output with "AOE=1" and set the analog output to velocity using "AO=V". The values of the control constants should all be set to their default values before starting to tune the motor. In other words, make sure that

KP=10 KD=500 KI=0 KA=0 FT=1

Now, to start tuning, the following program (next page) should be entered on the terminal. This gives the motor a series of step inputs that move the motor back and forth. You should be able to use the value given for the step input if the following error bound is set to a greater value. For example, in this case, it is desirable to set FB to 2000.

```
    DE1                 (define program 1)
    STM1=0.5            (load timer 1 with start time of 0.5 seconds and start timer 1)
1   SI:1000             (apply step input of 1000 pulses)
    WTTM1               (wait for timer 1 to count down to 0 seconds)
    SI:-1000            (apply step input of -1000 pulses)
    WTTM1               (wait for timer 1 to count down to 0 seconds)
    GT1                 (unconditionally goto 1)
    ED                  (end program 1 and exit program editor)
```

Now, execute the program and view what is happening on the oscilloscope. You should see an overdamped response similar to Figure 1.

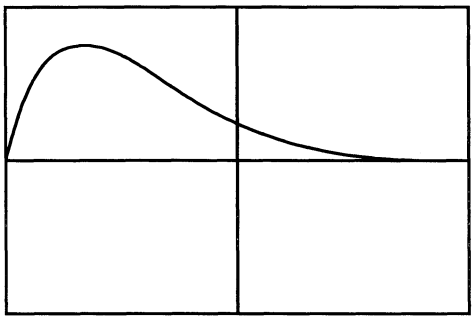


Figure 1
Overdamped response

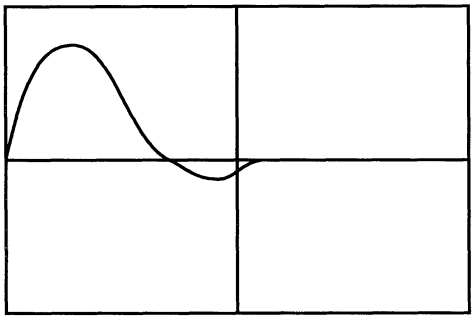


Figure 2
Less overdamped response

In order to improve the response, first work with the proportional control gain, KP. Increasing this will give a much crisper response to the step input. Beyond a point, however, the motor will start to overshoot (see figure 2), and increasing KP even further may result in oscillations (figure 3). You can get rid of overshoot by increasing the derivative control gain, KD. You will want a high value of KD in order to stabilize the system, but too large of a value of KD will set up a mechanical resonance in the system, which can be heard or felt. As a general rule, you should try to establish an upper limit on KD first, and then KP.

Next, try to work with the value for the integral control gain, KI. Start with KI=100, and increase it until you see the motor start to overshoot. A small amount of overshoot is needed in order for the system to perform well. Too large of a value for KI will cause too much overshoot. Again, see figure 2 for an illustration.

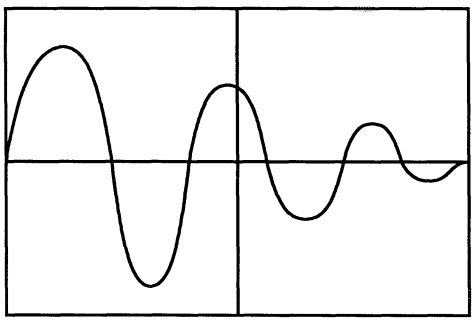


Figure 3
Oscillatory response

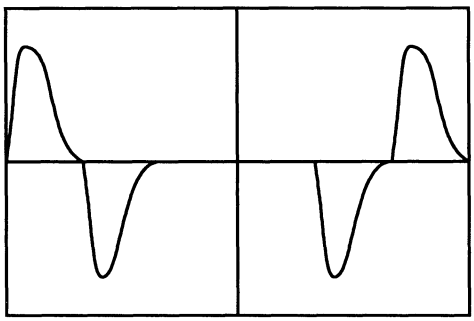


Figure 4
Following error response

To find a value for the filter time constant, FT, look at the motor and see if it settles when the axis is stopped. If not, the filter time constant should be increased. In general, the filter time constant should be increased for a lower torque to inertia ratio. Note that FT has no affect on the step response of the motor.

Finally, to find a good value for the acceleration feedforward constant, KA, you can move the axis using the following program, making sure that you first set the analog output to following error ("AO=F"). Also, you should substitute for the AC value the maximum acceleration that you think your system can handle.

```
      AUR=1          (load axis unit ratio of 1 pulse/unit, i.e. units=pulses)
      AM=0           (load absolute move position of 0 pulses)
      IM=40000       (load incremental move position of 40,000 pulses)
      VL=40000       (load velocity of 40,000 pulses/sec)
      JK=0           (load jerk percentage of 0%)
      AC=1000000     (load acceleration of 1,000,000 pulses/sec^2)
      DE1            (define program 1)
1     RIN            (run to incremental move position)
      WT IP          (wait for axis to be in position)
      RAN            (run to absolute move position)
      WT IP          (wait for axis to be in position)
      GT1            (unconditionally goto 1)
      ED             (end program 1 and exit program editor)
```

Start with KA=0. You should see something similar to figure 4 on the scope. Then, increase the value of KA in order to minimize the following error as much as possible (figure 5). Too large of a value of KA will cause the following error to increase, but with a change in phase, so that the positive voltages become negative and vice versa (figure 6).

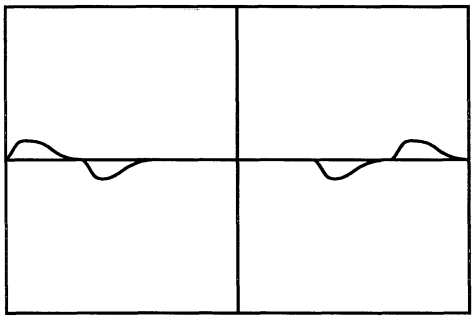


Figure 5
Following error with KA increased

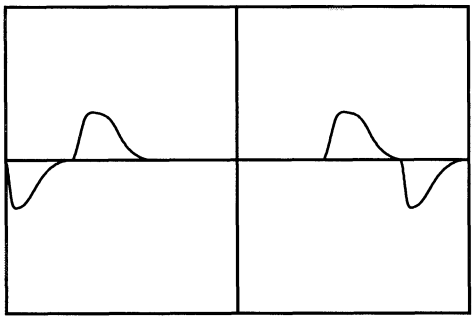


Figure 6
Following error with KA too large

c. Stepper Motor Configuration

This section describes:

1. how to set the current settings;
2. how to set up position feedback; and
3. how to use the position feedback constants.

1. Current settings (CC, SC)

If you are using a stepper motor, CC and SC should be set. The continuous current in percent of the continuous current rating of the drive is set by CC, and the power save current percentage is set by SC. The power save current percentage is the percentage of the continuous current, CC, that the stepper motor will drop to when the motor is not moving.

2. *Position Feedback (PFE, PF)*

When using a stepper motor, it is possible to use position feedback by setting PFE to 1. This enables position feedback. When position feedback is disabled, the axis position, AP, is set to the command position, CP. When position feedback is enabled, the axis position, AP, is set to the position feedback input, and the axis feedback resolution should be set to the appropriate value based on the feedback type used. The type of feedback is assigned by the PF command. Two different types can be implemented by the PF command:

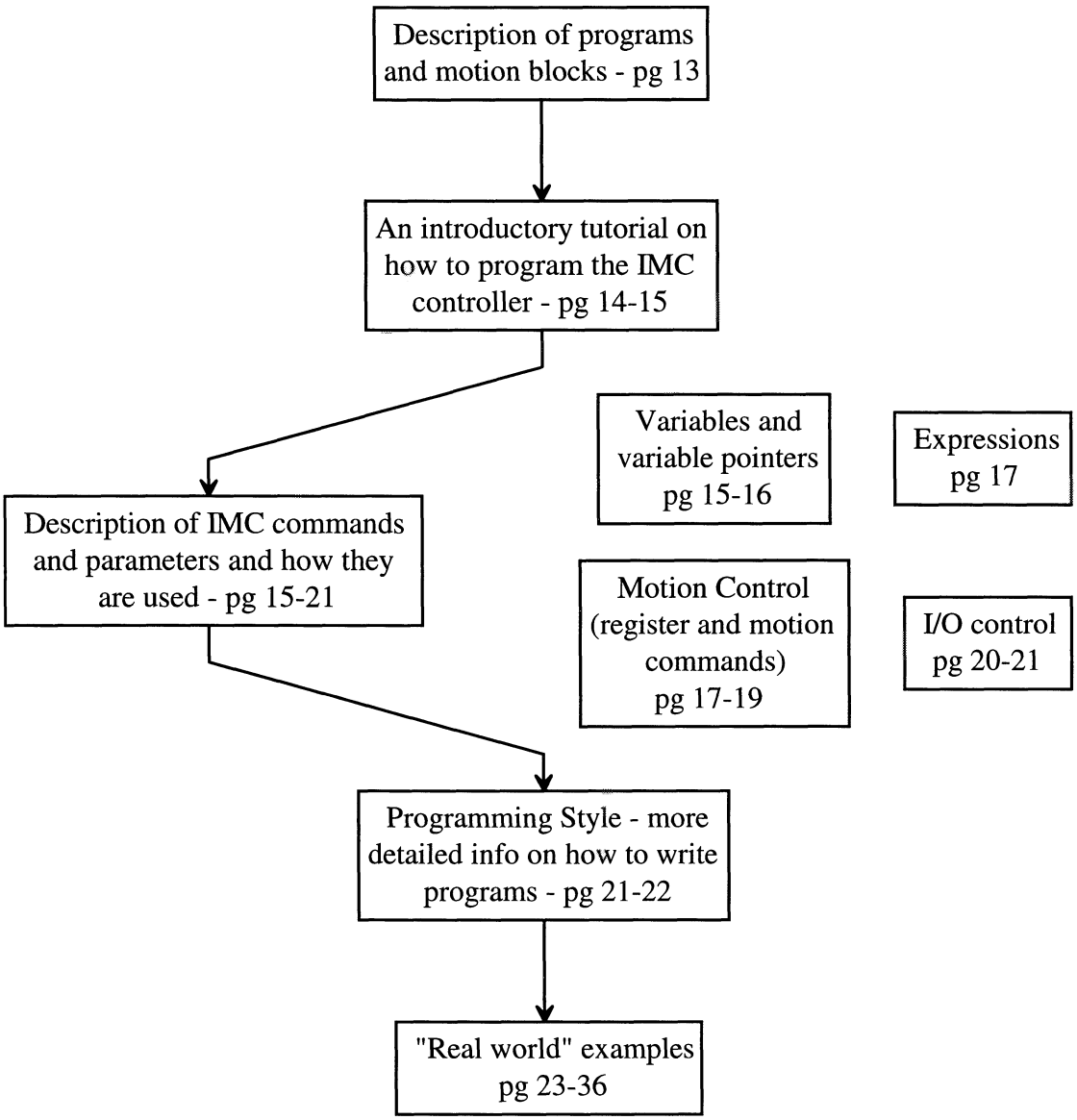
- pulse/direction (PD)
- quadrature (Q1, Q2, Q4)

For more information, see the PF command in the command summary.

3. *Position Feedback Constants (CT, BL, DB)*

When using position feedback, there are some constants that are used in the system. These are the correction time constant, the backlash constant, and the deadband constant. The correction time constant, CT, is the time that the controller waits between position corrections. The backlash constant, BL, is used to compensate for mechanical backlash in the system. Both of these together are used to make corrections once the stepper is outside the deadband. Finally, the deadband constant, DB, is the amount of position error allowed before the controller corrects the position error. Note that these constants are not used when position feedback is not used.

2. Programming



2. Programming

Introduction

Now that you have your IMC unit completely set up, you can now begin the process of telling the controller what you want it to do for your specific application. In order to "talk" to the controller, you must learn to speak in its language. The language of the IMC consists of commands that are powerful and easy to use. These commands tell the controller to do a specific action, such as loading the number 5 into a integer variable, moving the axis in the forward or reverse direction, or "turning on" one of the digital outputs. Combining commands like these with control flow statements such as GT ("goto") and GS ("gosub") will give you a program. Programs are entered into the program editor, and then may be executed by using the EX command. To execute specific motions, especially multiple speed moves, you may combine motion commands along with other possible commands into what is called a motion block. In this chapter, we will discuss how to "tell the controller what to do"; in other words, how to program the controller using programs and motion blocks and executing them.

A. Program/memory structure

There is 32K of memory allocated for storage of up to 4 programs and 100 motion blocks. These programs and motion blocks are all referred to by number, for example, program 1 to program 4 or motion block 1 to motion block 100. All of the programs can be executed concurrently, due to the multitasking capability of the controller. Alternatively, a program can be locked to the interpreter with the LK command so that no other program can execute along with it. Program 4 is reserved as the fault program. This means that it will execute anytime a fault occurs including the power failure fault, which occurs on power up.

Motion blocks can be executed like programs, but they cannot be executed concurrently with other motion blocks. In addition, only one motion block can be executed concurrently with the executing program(s). Motion blocks are like motion commands in that once they are issued, they override the currently executing motion block or motion. Motion blocks are mainly used to define complex motions, such as multiple speed moves, and also to define any other motion that you wish to be executed. Commands in motion blocks are executed in the proper spatial relationship which is set by the motion block. This means that outputs can be set or reset at a particular spot in a motion block by placing the set (STD, STDA) or reset (RSD, RSDA) command in the proper place in the motion block.

B. Program command summary

mnemonic	*description*
DE	defines program
DEL	deletes current statement in program/motion block editor
L	makes last statement the current statement in program/motion block editor
X	steps through program/motion block
LB	makes statement at label the current statement in the program editor
!	exits program/motion block editor
FLT	enters program/motion block editor and makes statement which faulted system current statement
/	comment
ED	ends program and exits program editor
GT	unconditionally "gotos" label
GS	unconditionally "gosubs" label
RET	returns from "gosub"
POP	pops "gosub" address from top of "gosub" stack
RST	resets "gosub" stack to empty
EX	executes program
KL	kills program
KLA	kills all programs

SUP	suspends motion
RSM	resumes motion
LK	locks interpreter to program
ULK	unlocks interpreter from program
EXSV	executes command stored in string variable
MB	defines motion block
MED	ends motion block and exits motion block editor
MEX	executes motion block
MRP	repeats motion from beginning of motion block
STM	loads start time of timer and starts timer
WT	waits for expression to be true
WT...ON...GT	waits for expression to be true or on condition becoming true "gotos" label
IF...GT	conditionally "gotos" label
IF...GS	conditionally "gosubs" label
STB...GT	sets boolean variable and if variable wasn't set "gotos" label

C. Starting to Program

This section gives an introductory tutorial on how to program the controller. Its purpose is to help you get started writing useful programs without concentrating on details. It is not intended to give a complete guide on how to program using programs and motion blocks, but to give you a good "feel" for how to enter and edit programs that will make the system work for your application.

As said before, an IMC program consists of commands that are entered in the program editor. The program editor is entered by using the DE command. The syntax of the DE command contains a parameter which is the number of the program you wish to start entering or editing. For example, if "DE1" is entered while not in the editor, the program editor will be entered and the first line of program 1 will be displayed. If program 1 does not exist, the first and only line of the program will be "ED", which defines the end of the program. Once the program editor is entered using the DE command, you can start to enter or edit a program.

The following example is a short program that executes a simple motion. This program, once executed, will set the axis position register to 0, load the appropriate register values and run the axis 12 units in the forward direction. We have added comments in parentheses so that each program statement is explained.

DE1	(define program 1)
SAP:0	(set axis position register to 0)
VL=10	(load velocity of 10 units/sec)
AC=40	(load acceleration rate of 40 units/sec^2)
AM=12	(load absolute move position of 12 units)
RAN	(run to absolute move position)
ED	(end program 1 and exit program editor)

Once you have entered the program, you can either execute it using the EX command, or edit it by entering the program editor using the DE command. Here you would enter "EX1" to execute it and "DE1" to enter the program editor and display line 1 of program 1.

When editing a program, you can choose from a number of commands that will aid in editing the program. The X command will advance to the next program line. If an integer *n* is given after X, the editor will advance through the next *n*-1 program lines to the *n*th line after the line that you were previously at. For example, if you were on the third line of the program and you entered "X 2", the editor would advance through the next line to the second line after the third line, which is the fifth line:

```
* AC=40
X 2
* AM=12
  RAN
```

In addition, the L command is used to go to the last program statement, which means the one immediately before the current statement. The LB command is used to go to a specific program label (see "Programming Style" for more information).

You can also delete the current statement in the editor by using the DEL command. This will also make the next statement the current statement. If you wish to enter a command in the place of the deleted one, do it while the next statement is the current statement. This will place the command in the statement before the current statement. Finally, the ! command is used to exit the program editor.

Motion blocks can be similarly entered and edited, except that labels are not used. Therefore, the LB command cannot be used in the motion block editor. The MB command enters the motion block editor at the first line of the motion block specified, and the MED command ends the motion block and exits the motion block editor.

D. Commands and Parameters

This section describes how to use the commands of the IMC and their parameters to make the system function for your specific application. The following material will be covered:

1. variables and variable pointers;
2. boolean, variable and string expressions;
3. motion control using register and motion commands; and
4. using I/O commands to control input and output to the controller.

1. Variables and Variable Pointers

This section is an introduction of the types of variables available in the controller. It also tells about using variable pointers as a means of referencing variables. First, a command summary is given for the variable commands. Then, the following topics are covered:

a. boolean, integer, floating point and string variables and
b. integer, floating point and string variable pointers.

Variable Command Summary

mnemonic	*description*
RSB	resets boolean variable
STB	sets boolean variable
BV	loads boolean variable
IV	loads integer variable
IVP	loads integer variable pointed to by another integer variable
DV	loads double integer variable with double expression
FV	loads floating point variable
FVP	loads floating point variable pointed to by integer variable
SV	loads string variable

a. Variables (BV*n*, IV*n*, DV*n*, FV*n*, SV*n*)

There are four types of variables that are used in the controller; they are boolean, floating point, integer, and string variables.

Boolean variables (BV*n*) can assume a value of 0 or 1, and are used mainly in conditional statements such as IF...GT (conditional goto) and WT (wait for expression to be true). They are also used in commands that enable something such as the digital outputs (DOE) or the computer interface format (CIE).

Integer variables (IV*n*) can have any value between -2,147,483,648 and 2,147,483,647. They are used mainly in variable expressions (see Section 2, "Expressions") and to load parameters. If, when doing calculations

with integer variables, you need to do a calculation outside of the range of integer variables, you can use a double integer variable expression (DV*n*). For an example of this, see DV in the command summary.

Floating point variables can assume any value between 1.5 x 10^{-39} (absolute value) to 1.7 x 10^{38} (absolute value) with up to nine digits precision, as well as 0. Floating point variables are used in variable expressions and to load parameters.

String variables can be loaded with a string that can be up to 127 characters long. String variables are used mainly in input/output commands such as GET, IN and OUT as a means of user interface.

b. Variable Pointers (IVP*n*, FVP*n*, SVIV*n*, SVIVP*n*)

Variable pointers are used to reference variables, which were covered in the previous section. A pointer contains the number of the variable that you wish to "point to". For example, if we want to have a pointer access integer variable 10, we can set any other integer variable, such as integer variable 1, to 10. Then, integer variable 1 (IV1) will be loaded with the pointer 10, which can be used to reference integer variable 10 (IV10). Suppose that IV10 is equal to 52. We now have

```
IV1=10
IV10=52
```

In order to reference IV10 with the pointer loaded into IV1, we use IVP1. The value of IVP1 is defined as the integer variable pointed to by integer variable 1. So, in this case, it would be integer variable 10. To summarize, we have

```
IV1=10
IVP1=IV10=52
```

One example of using pointers is sending a long list of characters to the display whose ASCII values are stored in integer variables. Suppose we have ASCII codes stored in integer variables 100 through 200. One could send them to the display using the ASC command one hundred times over:

```
ASCIV100
ASCIV101
...
ASCIV200
```

However, this may be time consuming. An alternate way of doing this uses variable pointers:

```
    IV1=100
1   ASCIVP1
    IV1=IV1+1
    IF IV1<=200 GT1
```

Here, IV1 is loaded with the pointer 100. Then, "ASCIVP1" will send the ASCII character stored in IVP1 to the display, where IVP1 is equal to the integer variable pointed to by IV1. So, this statement is equivalent to "ASCIV100" the first time around. Then IV1 is incremented by 1 and checked to see if it is less than or equal to 200. Then, it goes back to the statement at label 1, which sends ASCII character IVP1 to the display. This is now equivalent to saying "ASCIV101". Then IV1 is again incremented by 1 and checked to see if it is less than or equal to 200, and so on until IV1 is equal to 200. At this point, the program will execute "ASCIVP1", which would be equivalent to "ASCIV200". Then IV1 is incremented by 1, and therefore equal to 201. Now, when IV1 is checked to be less than or equal to 200, it fails the check and so goes on to the next program line.

So, variable pointers are used as a means of indirectly accessing variables. And as shown in the above example, programs can be much shorter when using variable pointers.

2. Expressions

In most of the commands that you use, the parameter that is part of the command's syntax will be either a boolean expression (bexpr), variable expression (vexpr) or string expression (sexpr). These expressions associate operands, which can be variables or parameters, and operators to produce a meaningful "expression" which takes on a certain value. This value is loaded into the variable or parameter on the left hand side of the assignment operator.

Conversion between variable types (boolean, integer, floating point, string) is done automatically. This means that all operands in a command statement will be the same type as the variable or parameter which will be loaded with the result. If the intermediate calculation must be done in a different format, then an additional command statement should be used.

When using the command summary in Appendix A, you will see that for many commands, there are parameters associated with them, and most cases they will be "bexpr" or "vexpr". Only the SV command uses "sexpr" as a parameter. Use Appendix B, "Expression Charts", to see which variables and other operands and operators are used in these expressions.

3. Motion Control

This section will show how register and motion commands are used to achieve motion control. The following topics will be covered:

a. setting the motion registers to define the position, velocity, acceleration, and jerk of the IMC axis and
b. using motion commands to move the axis.

a. Register Commands

This section describes the use of register commands to define the position, velocity, acceleration, and jerk of the axis. First, the register command summary is given. Then, the following are explained:

1. how to define the position and destination of the axis;
2. the motion parameter;
3. how to define the velocity and acceleration of the axis;
4. how to define the jerk percentage of the axis; and
5. how to define the acceleration percentage and move time of the axis.

Register Command Summary

mnemonic	*description*
AM	loads absolute move position
OM	loads offset move position
IM	loads incremental move position
MP	assigns motion parameter
JK	loads jerk percentage
AC	loads acceleration & deceleration
DC	loads deceleration
VL	loads velocity
ACP	loads acceleration & deceleration percentage
DCP	loads deceleration percentage
MVT	loads move time
SAP	sets axis position register
SOP	sets offset position register
IOP	increments offset position register

1. Axis Position (SAP, SOP, IOP) and Destination (AM, OM, IM)

The position of the axis is defined by the axis position register, and is set by using the SAP command. Note that setting the axis position register cannot be done within a motion block. There is also another level of defining the axis position, which is called the offset position. This is set using the SOP command, and can be incremented using the IOP command. Note that these commands can be used in motion blocks.

In order to define a motion, we must also define the destination of the axis. In other words, we must define not only the current position, but also to where the axis will move. Hence, the IMC has three commands which define specific types of moves:

- absolute move (AM) - the position that the axis will move to with respect to the zero axis position;
- offset move (OM) - the position that the axis will move to with respect to the zero offset position; and
- incremental move (IM) - the position that the axis will move to with respect to the current axis position, AP. In other words, this is the distance that the motor will move from the current position.

These commands are directly linked to the run commands, which move the axis according to these positions. For more information, see the "Motion Commands" section.

2. Motion Parameter (MP)

The MP command is used to assign the motion parameter, which can be velocity (V) or time (T). The motion parameter defines which register commands can be used to define axis motion. If the motion parameter is velocity, the motion commands make use of VL, AC, DC and JK to define the motion profile. If the motion parameter is time, the motion commands make use of MVT, ACP, DCP, and JK to define the motion profile.

3. Velocity (VL) and Acceleration (AC, DC)

When using velocity as the motion parameter, you can define the velocity (VL) and acceleration (AC) of the axis. If necessary, you can also define the deceleration (DC) of the axis. Note that if DC is not specified, the deceleration of the axis will be equal to the acceleration of the axis. See the command summary for more information.

4. Jerk Percentage (JK)

The jerk percentage is defined as the percentage of acceleration/deceleration time that the axis will jerk. When JK is set to 0, there is no jerk limit, i.e. the jerk is infinite. Limiting the jerk on the axis is helpful in applications where you do not want the axis to immediately achieve an acceleration, such as starting up and stopping a conveyor belt. For an example of how JK is used, see examples 1, 2, 4, and 6 in section F of Programming.

5. Move Time (MVT) and Acceleration Percentage (ACP, DCP)

When using time as the motion parameter, you can define the move time (MVT) and acceleration percentage (ACP) of the axis. The acceleration percentage is defined as the percentage of move time that the axis will accelerate. If necessary, you can also define the deceleration percentage (DCP) of the axis. Note that if DCP is not specified, the deceleration percentage of the axis will be equal to the acceleration percentage of the axis if it is not greater than 50%. See the command summary for more information.

b. Motion Commands

This section describes the use of motion commands to move the axis. First, the motion command summary is given. Then, the following are explained:

1. the run, stop and halt commands and
2. the home commands.

Motion Command Summary

mnemonic	*description*
HT	halts all motion
ST	stops all motion
RAN	runs to absolute move position
RON	runs to offset move position
RIN	runs to incremental move position
RFN	runs forward
RRN	runs reverse
HFH	homes forward to home input
HRH	homes reverse to home input
HFO	homes forward to overtravel input
HRO	homes reverse to overtravel input
HFM	homes forward to marker
HRM	homes reverse to marker
SI	applies step input

1. Run (RAN, RON, RIN, RFN, RRN), Stop (ST) and Halt (HT) Commands

The run commands cause the axis to either run to a specified position, or to run indefinitely until another run command or a stop or halt command is issued. If the run commands are not interrupted, they will either execute a standard trapezoidal or triangular motion profile to move from the current position to the specified position (RAN, RON, RIN), or run indefinitely (RFN, RRN).

There are basically two types of run commands: those that use the position registers and move parameters (RAN, RON, RIN) and those that do not (RFN, RRN). Here is a brief explanation of each:

- RAN - this command runs the axis to the absolute move position, AM.
- RON - this command runs the axis to the offset move position, OM.
- RIN - this command runs the axis to the incremental move position, IM.
- RFN - this command runs the axis indefinitely in the forward direction.
- RRN - this command runs the axis indefinitely in the reverse direction.

The stop command, ST, decelerates the axis with the deceleration loaded to a complete stop. The halt command, HT, immediately halts the axis. The halt command should only be used in extreme situations. See the command summary for more information.

2. Home Commands (HFH, HRH, HFO, HRO, HFM, HRM)

The home commands are used to home the axis using four different inputs: the home input (HFH, HRH), the forward overtravel input (HFO), the reverse overtravel input (HRO), and the marker input (HFM, HRM). They will run the axis at a fixed velocity of 4,096 pulses/sec until the input is encountered. For more information, see the command summary.

4. I/O Control

This section will show how input/output commands are used to control the input/output of the IMC unit. First, a command summary is given for the input/output commands. Then, the following are explained:

a. digital input filtering;
b. using the digital output fault;
c. enabling the digital and analog outputs; and
d. data input/output with and without the display.

For information about using digital inputs and outputs and set points, see "Analog & Digital I/O" in the Hardware section of the IMC Controller Setup.

Input/Output Command Summary

mnemonic	*description*
RSD	resets digital output
RSDA	resets all digital outputs
STD	sets digital output
STDA	sets all digital outputs
TGD	toggles digital output
DO	sets and resets digital outputs according to number
DFT	loads digital input filter time
DIF	assigns filter to digital input
OFE	enables fault on output fault
DOE	enables digital outputs
AOE	enables analog output
AO	loads analog output with number/assigns signal to analog output
SPA*n*B	loads set point A beginning
SPB*n*B	loads set point B beginning
SPA*n*E	loads set point A ending
SPB*n*E	loads set point B ending
SPA*n*F	forgets set point A
SPB*n*F	forgets set point B
PRT	prints string to display with carriage return and line feed
PNT	prints string to display without carriage return and line feed
ASC	sends ASCII code to display
GET	gets one character from key buffer
IN	inputs variable value from key buffer
OUT	outputs variable value to display
FW	assigns field width of output variable on display
DP	assigns number of places after decimal point on display

a. Digital Input Filtering (DIF, DFT)

The digital inputs can be filtered using the DIF and DFT commands, in order to deal with problems such as contact bounce. The DIF command defines which digital inputs are to be filtered, and the DFT command sets a digital input filter time. The digital input filter time is defined as the minimum duration of a pulse that the filter will pass.

b. Digital Output Fault (OFE)

If you wish to have the controller fault on a digital output fault, you must set OFE equal to 1. Also note that a fault on a digital output is cleared when the output is turned off.

c. Digital and Analog Output Enabling (DOE, AOE)

When the controller is powered up, both the digital outputs and the analog output are disabled. In order to enable them, you must use the DOE and AOE commands. To enable the digital outputs, set DOE to 1, and to enable the analog output, set AOE to 1.

d. Data Input/Output (PRT, PNT, ASC, GET, IN, OUT, FW, DP)

This section describes how user data is input and output by the controller. Data is input using the IN and GET commands. User data can be output to the display or the terminal using the PRT, PNT, ASC and OUT commands. When using the OUT command, it is displayed with a certain format which is set by the controller and the FW and DP commands.

The PRT and PNT commands print character strings to the display or terminal, depending on the setting of DSE, the display format enable. PRT prints a string with a carriage return and line feed, and PNT prints a string without a carriage return and line feed.

The ASC command sends an ASCII code to the display or terminal. This command is almost always used with the display enabled. The operations of the display are executed by certain ASCII codes described in Chapter 4, "Using the Display".

The IN command inputs a variable value from the key buffer. The GET command does the same thing, but only takes one character from the key buffer. So, the GET command can be used to handle single keystrokes, for example.

The OUT command outputs a variable value to the display or terminal, depending on the setting of DSE. When a number is set to a string variable, then it is put in the string variable in standard form.

Standard form for floating point numbers is -d.ddddddddE-dd. The number is displayed with at most nine digits. If more digits would be required to display the number, then an exponent is displayed. Otherwise, the exponent is suppressed since it is zero. The number is always displayed to maximum precision. For example, 1.1E-8 is displayed as 0.000000011 but 1.1E-9 is displayed as 1.1E-9.

When an OUT statement is used, then the number is printed using the field width (FW) and decimal places (DP) parameters. If a number will not fit in the specified field width, then the controller prints asterisks (*) in the field. Numbers are always right justified in the field and all specified decimal places are printed, even if they are zero.

When OUT is used, integer variables are always printed in the integer format and floating point variables are always printed in the floating point format. The exponent form is never used for integer variables. When E is printed, it always takes up three places, for example, E12 or E01. Minus exponents will never be printed since numbers less than the last displayed decimal place in magnitude are displayed as zero.

E. Programming Style/Control Flow

Now that you know about many of the useful commands of the IMC, you can combine them with control flow statements such as GT (goto) and GS (gosub) to give you a program. This section describes these control flow statements, and in doing so, shows how to write programs for the IMC. The following topics are covered:

1. Unconditional gotos and gosubs;
2. Conditional statements; and
3. Manipulating the gosub stack.

Refer to Section B, "Program Command Summary", for a listing of the program commands. For more information, see the complete command summary in Appendix A. For practical examples of IMC programs, see Section F, "Programming Examples".

1. Unconditional Gotos and Gosubs (GT, GS, RET)

The GT command is defined as an unconditional goto. Unconditional gotos are used literally to go to any other program statement in the currently executing program. These program statements have labels associated with them. A label is an integer number from 1 to 999 that comes immediately before a program statement. Labels are used in programs as a means of referencing a program statement, much like the way that pointers are used to reference variables. So, the program statement "GT 10", once executed by the controller, will go to the program statement with the label "10" immediately before it.

The GS command is defined as an unconditional gosub. It is used to goto a subroutine. A subroutine is a section of a program devoted to a routine that can be accessed multiple times by the GS command. GS is used in the same way as GT, but once the program goes to the statement, it expects the RET command. The RET command returns to the statement immediately following the gosub.

For more information about and examples of how these commands are used, see the command summary.

2. Conditional Statements (IF...GT, IF...GS, WT, WT...ON...GT, STB...GT)

Conditional statements are used to test the controller for a certain condition, such as the axis being in position (IP) or integer variable 5 being equal to 7 (IV5=7). Conditions are represented by boolean expressions (bexpr). If or when a condition is satisfied, the controller will then either goto a program statement or goto a subroutine. The following is a list of the different kinds of conditional statements available: WT, WT...ON...GT and STB...GT.

- The IF...GT command is a conditional goto. If the condition between the IF and GT is satisfied, then the program goes to the label specified.
- The IF...GS command is a conditional gosub. If the condition between the IF and GS is satisfied, then the program goes to the subroutine beginning at the label specified.
- The WT command waits for the condition specified to be satisfied. When it is satisfied, it continues execution of the program.
- The WT...ON...GT command waits for the condition between the WT and ON to be satisfied, or the condition between the ON and GT being satisfied, it goes to the label specified. If the condition between WT and ON is satisfied before the other one, the program continues execution.
- The STB...GT command sets the boolean variable specified to 1. If the boolean variable was not set to 1 before this statement was executed, it goes to the label specified.

Again, for more information about and examples of how these commands are used, see the command summary.

3. Gosub Stack Manipulation (POP, RST)

The gosub stack is basically a list of labels in the controller. These are the labels of the statements of the subroutines that are currently executing. There is a different stack for each currently executing program. If a stack overflows or underflows due to too many nested gosubs or improper use of gosubs, then the system faults and the offending program is killed.

One way that the gosub stack can be manipulated is to have a command that will remove the label of the last executed gosub from the stack. This command is the POP command. It is used to exit a subroutine without returning. Also, the RST command can be used, but this will remove all labels from the gosub stack. Therefore the RST command can be used to exit a subroutine if there are nested gosubs and you wish to go back to normal execution.

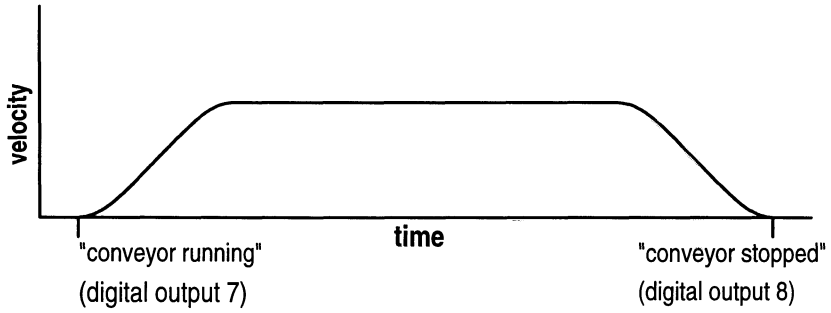
Again, for more information about and examples of how these commands are used, see the command summary.

F. Programming Examples

1. Accurate Velocity Control (conveyor)

Servo and stepping motors are necessary in many applications requiring accurate velocity control. Unlike conventional variable speed motors, Whedco servo and stepping motor systems provide long-term velocity error of less than a tenth of a percent. Tighter process constraints imposed on many system designers make this a mandatory feature in many velocity control applications. In addition, Whedco servo and stepping motor systems operate at higher speeds than typical d.c. motors. Many Whedco systems are available to operate at speeds in excess of 5000 r.p.m.. Finally, Whedco servo and stepping motor systems operate at a constant speed regardless of input line voltage, a critical feature in maintaining accurate velocity control.

The example below shows how this feature can be used in a conveyor system.

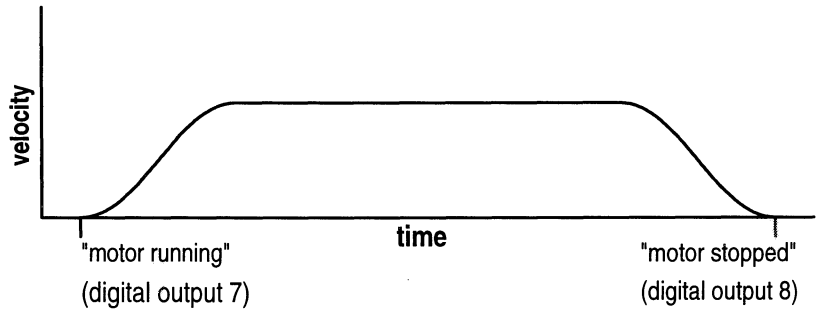



```
                    PWE=1                   (enable position register wrap)

Program 1           DE1                     (define program 1)
(main           1   IF EG1 GS10             (conditionally gosub 10 if positive edge sensitive digital input 1, i.e.
program)                                     the "run conveyor" input, is active)
                    IF EG2 GS20             (conditionally gosub 20 if positive edge sensitive digital input 2, i.e.
                                             the "stop conveyor" input, is active)
                    GT1                     (unconditionally goto 1)
                10  MEX1                    (execute motion block 1, which is the "run conveyor" motion block)
                    RET                     (return from gosub)
                20  MEX2                    (execute motion block 2, which is the "stop conveyor" motion block)
                    RET                     (return from gosub)
                    ED                      (end program 1 and exit program editor)

Motion Block 1      MB1                     (define motion block 1)
("run conveyor")    RSD8                    (reset digital output 8, i.e. turn the "conveyor stopped" output off)
                    STD7                    (set digital output 7, i.e. turn the "conveyor running" output on)
                    VL=2                    (load velocity of 2 units/sec)
                    AC=50                   (load acceleration of 50 units/sec^2)
                    JK=25                   (load jerk percentage of 25%)
                    RFN                     (run forward)
                    MED                     (end motion block 1 and exit motion block editor)

Motion Block 2      MB2                     (define motion block 2)
("stop conveyor")   DC=200                  (load deceleration of 200 units/sec^2)
                    JK=25                   (load jerk percentage of 25%)
                    ST                      (stop all motion)
                    RSD7                    (reset digital output 7, i.e. turn the "conveyor running" output off)
                    STD8                    (set digital output 8, i.e. turn the "conveyor stopped" output on)
```

```
MED                    (end motion block 2 and exit motion block editor)
```

2. Point-to-Point Positioning

Numerous applications require point-to-point position control where a relative or absolute position destination can be specified. The C-Version Independent Motion Controllers offer many possibilities for this type of move. Position moves can be preprogrammed and executed in response to a discrete input. Alternatively, destination position can be conditional based on program variables or external commands received from a host computer. The example below shows the use of encoded digital inputs to achieve specific axis motions.




```
Program 1 (main program)
        DE1                 (define program 1)
    1   WT EG6              (wait for positive edge sensitive digital input 6, i.e. the "execute
                             motion" input, to be active)
        IV1=EG AND 31       (load integer variable 1 with the result of EG AND 31, i.e. with the
                             integer value of encoded positive edge sensitive digital inputs 1
                             through 5; note that 31(10)=000000011111(2))
        IF IV1=0 GT1        (conditionally goto 1 if IV1 equals 0)
        IF IV1>9 GT1        (conditionally goto 1 if IV1 is greater than 9)
        IF IV1>5 GT10       (conditionally goto 10 if IV1 is greater than 5)
        IF MB GT1           (conditionally goto 1 if a motion block is executing)
        RSD8                (reset digital output 8, i.e. turn the "motor in position" output off)
        STD7                (set digital output 7, i.e. turn the "motor running" output on)
        MEXIV1              (execute motion block IV1, which can only be 1 through 5)
        GT1                 (unconditionally goto 1)
   10   EX2                 (execute program 2)
        GT1                 (unconditionally goto 1)
        ED                  (end program 1 and exit program editor)

Program 2
        DE2                 (define program 2)
        GT IV1              (unconditionally goto IV1, which can only be 6 through 9)
    6   SUP                 (suspend motion)
        WT IP               (wait for axis to be in position)
        RSD7                (reset digital output 7, i.e. turn the "motor running" output off)
        GT10                (unconditionally goto 10)
    7   RSM                 (resume motion)
        IF N SRC GT10       (conditionally goto 10 if status register bit 12 is 0, i.e. if the motion
                             generator is not enabled)
        STD7                (set digital output 7, i.e. turn the "motor running" output on)
        GT10                (unconditionally goto 10)
```

```
8  DC=100        (load deceleration of 100 units/sec²)
   JK=25         (load jerk percentage of 25%)
   ST            (stop all motion)
   WT IP         (wait for axis to be in position)
   RSD7          (reset digital output 7, i.e. turn the "motor running" output off)
   GT10          (unconditionally goto 10)
9  RSD8          (reset digital output 8, i.e. turn the "motor in position" output off)
   STD7          (set digital output 7, i.e. turn the "motor running" output on)
   HFM           (home forward to marker)
   WT IP         (wait for motor to be in position)
   SAP:0         (set absolute position register equal to 0)
   RSD7          (reset digital output 7, i.e. turn the "motor running" output off)
   STD8          (set digital output 8, i.e. turn the "motor in position" output on)
10 ED            (end program 2 and exit program editor)
```

Motion Block 1

```
MB1              (define motion block 1)
VL=5             (load velocity of 5 units/sec)
AC=40            (load acceleration of 40 units/sec²)
JK=25            (load jerk percentage of 25%)
AM=10            (load absolute move position of 10 units)
RAN              (run to absolute move position)
RSD7             (reset digital output 7, i.e. turn the "motor running" output off)
STD8             (set digital output 8, i.e. turn the "motor in position" output on)
MED              (end motion block 1 and exit motion block editor)
```

Motion Block 2

```
MB2              (define motion block 2)
VL=5             (load velocity of 5 units/sec)
AC=40            (load acceleration of 40 units/sec²)
JK=25            (load jerk percentage of 25%)
IM=5.5           (load incremental move position of 5.5 units)
RIN              (run to incremental move position, i.e. run to current position + 5.5
                  units)
RSD7             (reset digital output 7, i.e. turn the "motor running" output off)
STD8             (set digital output 8, i.e. turn the "motor in position" output on)
MED              (end motion block 2 and exit motion block editor)
```

Motion Block 3

```
MB3              (define motion block 3)
VL=5             (load velocity of 5 units/sec)
AC=40            (load acceleration of 40 units/sec²)
JK=25            (load jerk percentage of 25%)
AM=20            (load absolute move position of 20 units)
RAN              (run to absolute move position)
RSD7             (reset digital output 7, i.e. turn the "motor running" output off)
STD8             (set digital output 8, i.e. turn the "motor in position" output on)
MED              (end motion block 3 and exit motion block editor)
```

Motion Block 4

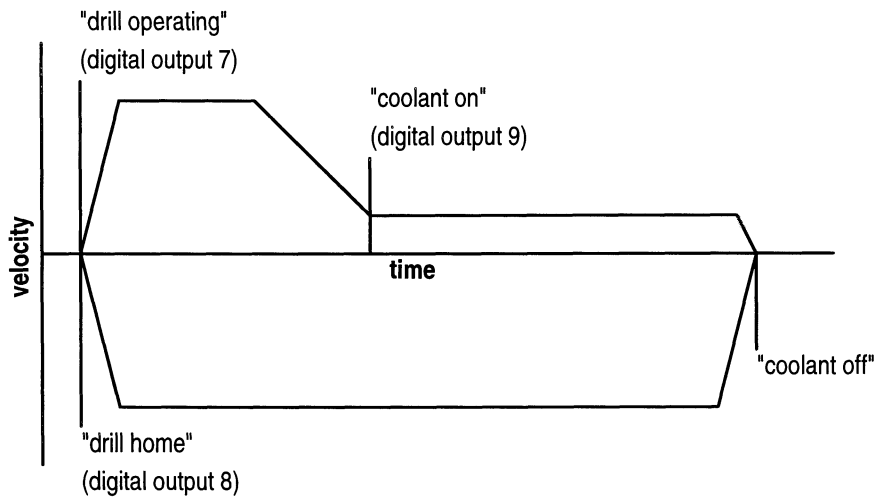
```
MB4              (define motion block 4)
VL=5             (load velocity of 5 units/sec)
AC=40            (load acceleration of 40 units/sec²)
JK=25            (load jerk percentage of 25%)
IM=4.5           (load incremental move position of 4.5 units)
RIN              (run to incremental move position, i.e. run to current position + 4.5
                  units)
RSD7             (reset digital output 7, i.e. turn the "motor running" output off)
```

	STD8	(set digital output 8, i.e. turn the "motor in position" output on)
	MED	(end motion block 4 and exit motion block editor)
Motion Block 5	MB5	(define motion block 5)
	VL=5	(load velocity of 5 units/sec)
	AC=40	(load acceleration of 40 units/sec^2)
	JK=25	(load jerk percentage of 25%)
	AM=0	(load absolute move position of 0 units)
	RAN	(run to absolute move position)
	RSD7	(reset digital output 7, i.e. turn the "motor running" output off)
	STD8	(set digital output 8, i.e. turn the "motor in position" output on)
	MED	(end motion block 5 and exit motion block editor)

3. Multiple Speed Position Moves (drilling)

In certain applications, different motor speeds are required when specific positions are achieved. This is common in applications such as drilling where a rapid infeed is followed by a slower drill-to-depth move. The C-Version Independent Motion Controllers include complete capabilities to define this type of move. Motion blocks allow a move to be configured where the exact behavior of the move can be described such that each segment of the move is completed without blending any of the move segments. Such moves can also include outputs which turn on as specific speeds and/or positions are achieved for auxiliary operations such as turning on coolant.

The example below demonstrates this feature in a drilling application. Motion block 1 defines the profile shown and also uses digital outputs, one which is used to turn on coolant.



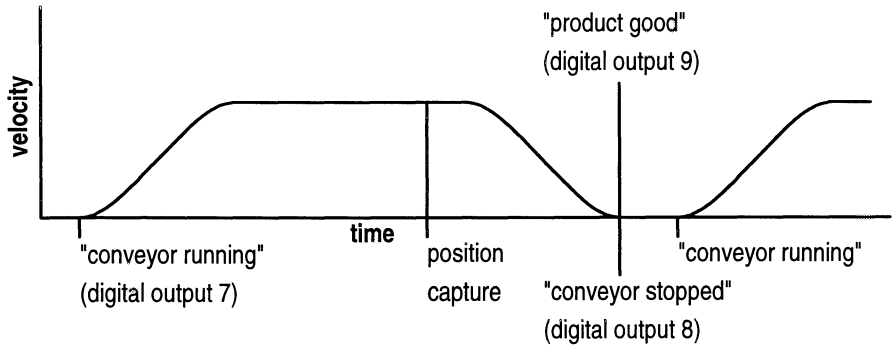
Program 1		DE1	(define program 1)
(main	1	IF EG1 AND N MB	
program)		GS10	(conditionally gosub 10 if positive edge sensitive digital input 1, i.e. the "run drill" input, is active and no motion block is executing)
		IF EG2 GS20	(conditionally gosub 20 if positive edge sensitive digital input 2, i.e. the "stop drill" input, is active)
		IF EG3 AND N MB	
		GS30	(conditionally gosub 30 if positive edge sensitive digital input 3, i.e. the "home drill" input, is active and no motion block is executing)

```
                GT1             (unconditionally goto 1)
             10 MEX1            (execute motion block 1, which is the "run drill" motion block)
                RET             (return from gosub)
             20 MEX2            (execute motion block 2, which is the "stop drill" motion block)
                RET             (return from gosub)
             30 MEX3            (execute motion block 3, which is the "home drill" motion block)
                RET             (return from gosub)
                ED              (end program 1 and exit program editor)

Motion Block 1  MB1             (define motion block 1)
("run drill")   RSD8            (reset digital output 8, i.e. turn the "drill home" output off)
                STD7            (set digital output 7, i.e. turn the "drill operating" output on)
                VL=20           (load velocity of 20 units/sec)
                AC=200          (load acceleration of 200 units/sec²)
                AM=15           (load absolute move position of 15 units)
                RAN             (run to absolute move position)
                VL=5            (load velocity of 5 units/sec)
                STD9            (set digital output 9, i.e. turn the "coolant on" output on)
                AC=40           (load acceleration of 40 units/sec²)
                AM=20           (load absolute move position of 20 units)
                RAN             (run to absolute move position)
                RSD9            (reset digital output 9, i.e. turn the "coolant on" output off)
                VL=20           (load velocity of 20 units/sec)
                AC=200          (load acceleration of 200 units/sec²)
                AM=0            (load absolute move position of 0 units)
                RAN             (run to absolute move position)
                RSD7            (reset digital output 7, turn the "drill operating" output off)
                STD8            (set digital output 8, turn the "drill home" output on)
                MED             (end motion block 1 and exit motion block editor)

Motion Block 2  MB2             (define motion block 2)
("stop drill")  DC=200          (load deceleration of 200 units/sec²)
                ST              (stop all motion)
                RSD9            (reset digital output 9, i.e. turn the "coolant on" output off)
                RSD7            (reset digital output 7, i.e. turn the "drill operating" output off)
                MED             (end motion block 2 and exit motion block editor)

Motion Block 3  MB3             (define motion block 3)
("home drill")  VL=20           (load velocity of 20 units/sec)
                AC=200          (load acceleration of 200 units/sec²)
                AM=0            (load absolute move position of 0 units)
                RAN             (run to absolute move position)
                RSD7            (reset digital output 7, turn the "drill operating" output off)
                STD8            (set digital output 8, turn the "drill home" output on)
                MED             (end motion block 3 and exit motion block editor)
```

4. High-speed Position Capture (index and cut)

The performance of many applications, in particular those requiring conditional indexing, can be improved by obtaining real-time position information. This is often difficult given inherent processing latencies in computer systems. The C-Version Independent Motion Controllers contain a dedicated high-speed input for a position sensor. This input will capture and store the position of the motor at the time this input is received in less than 500 nanoseconds.

The example below shows an application where the position is captured once a photo eye senses the product registration. Then, the motor runs forward for a specified number of units and stops, waiting for the next input. During this waiting time, the product is usually cut.



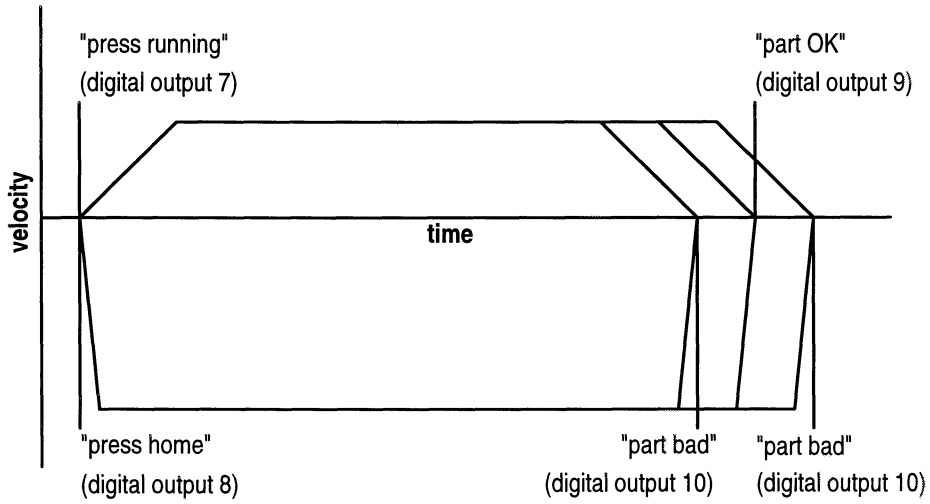
Program 1		DE1	(define program 1)
(main	1	IF EG1 GS10	(conditionally gosub 10 if positive edge sensitive digital input 1, i.e.
program)			the "run conveyor" input, is active)
		IF EG2 GS20	(conditionally gosub 20 if positive edge sensitive digital input 2, i.e.
			the "stop conveyor" input, is active)
		GT1	(unconditionally goto 1)
	10	EX2	(execute program 2, which is the "run conveyor" program)
		RET	(return from gosub)
	20	KL2	(kill program 2)
		MEX1	(execute motion block 1, which is the "stop conveyor" motion block)
		RET	(return from gosub)
		ED	(end program 1 and exit program editor)
Program 2		DE2	(define program 2)
("run conveyor")		SAP:0	(set actual position register equal to 0)
		DO:64	(set digital output 7, i.e. turn the "conveyor running" output on, and
			reset all other digital outputs; note that 64_{10}=000001000000_2)
		VL=2	(load velocity of 2 units/sec)
		AC=50	(load acceleration of 50 units/sec^2)
		JK=25	(load jerk percentage of 25%)
		AM=20	(load absolute move position of 20 units)
		RAN	(run to absolute move position)
		WT IOD ON IP GT10	(wait for I/O register bit 13 to be set, i.e. wait for capture input edge,
			or on the axis being in position goto 10)
		FV1=AT+5	(load floating point variable 1 with position capture plus 5 units,
			which is the separation between the photoeye and the cutter)
		AM=FV1	(load absolute move position of "FV1" units)
		RAN	(run to absolute move position)
		WT IP	(wait for axis to be in position)
		DO:384	(set digital output 8, i.e. turn the "conveyor stopped" output on, set
			digital output 9, i.e. turn the "product good" output on, and reset all
			other digital outputs; note that 384_{10}=000110000000_2)
		GT20	(unconditionally goto 20)
	10	DO:640	(set digital output 8, i.e. turn the "conveyor stopped" output on, set
			digital output 10, i.e. turn the "product out of registration" output on,
			and reset all other digital outputs; note that 640_{10}=001010000000_2)
	20	ED	(end program 2 and exit program editor)

Motion Block 1	MB1	(define motion block 1)
("stop conveyor")	DC=200	(load deceleration of 200 units/sec^2)
	JK=25	(load jerk percentage of 25%)
	ST	(stop all motion)
	RSD7	(reset digital output 7, i.e. turn the "conveyor running" output off)
	STD8	(set digital output 8, i.e. turn the "conveyor stopped" output on)
	MED	(end motion block 1 and exit motion block editor)

5. Torque-limited Moves (pressing)

In servo systems, not only can the speed and position of the motor be controlled but also the output torque. This is useful in applications where excess force can damage parts or where the assembly specification includes a force with which the part must be inserted. The Whedco Independent Motion Controllers allow the user to directly control the torque parameter in conjunction with position. This capability is not available in stepping motor systems.

The example below shows a pressing application, where the IMC controls the force applied by means of a torque limited-move.



Program 1		DE1	(define program 1)
(main	1	IF EG1 AND N BV1	
program)		GS10	(conditionally gosub 10 if positive edge sensitive digital input 1, i.e. the "run press" input, is active and boolean variable 1 is reset)
		IF EG2 GS20	(conditionally gosub 20 if positive edge sensitive digital input 2, i.e. the "stop press" input, is active)
		IF EG3 AND BV1	
		GS30	(conditionally gosub 30 if positive edge sensitive digital input 3, i.e. the "home press" input, is active and boolean variable 1 is set)
		GT1	(unconditionally goto 1)
	10	EX2	(execute program 2, which is the "run press" program)
		RET	(return from subroutine)
	20	KL2	(kill program 2)
		MEX1	(execute motion block 1, which is the "stop press" motion block)
		RET	(return from subroutine)
	30	MEX2	(execute motion block 2, which is the "home press" motion block)
		RET	(return from subroutine)

	ED	(end program 1 and exit program editor)
Program 2	DE2	(define program 2)
("run press")	DO:64	(set digital output 7, i.e. turn the "press running" output on, and reset all other digital outputs; note that $64_{10}=000001000000_2$)
	KI=0	(load integral control gain of 0)
	TC=50	(load torque limit current of 50% of continuous current rating of drive)
	TLE=1	(enable torque limit)
	VL=5	(load velocity of 5 units/sec)
	AC=40	(load acceleration of 40 units/sec^2)
	AM=20	(load absolute move position of 20 units)
	RAN	(run to absolute move position)
	WT TL ON N SRA0 GT10	(wait for axis to be at torque limit or on axis status register bit 0 becoming 0, i.e. the motion generator being disabled, goto 10)
	FV1=AP	(load floating point variable 1 with the actual position)
	IF FV1<18 GT10	(conditionally goto 10 if floating point variable 1 is less than 18)
	IF FV1>19 GT10	(conditionally goto 10 if floating point variable 1 is greater than 19)
	STD9	(set digital output 9, i.e. turn the "part OK" output on)
	GT20	(unconditionally goto 20)
10	STD10	(set digital output 10, i.e. turn the "part bad" output on)
20	MEX2	(execute motion block 2)
	ED	(end program 2 and exit program editor)
Motion Block 1	MB1	(define motion block 1)
("stop press")	DC=200	(load deceleration of 200 units/sec^2)
	ST	(stop all motion)
	RSD7	(reset digital output 7, i.e. turn the "press running" output off)
	STB1	(set boolean variable 1)
	MED	(end motion block 1 and exit motion block editor)
Motion Block 2	MB2	(define motion block 2)
("home press")	VL=20	(load velocity of 20 units/sec)
	AC=200	(load acceleration of 200 units/sec^2)
	AM=0	(load absolute move position of 0 units)
	RAN	(run to absolute move position)
	TLE=0	(disable torque limit)
	KI=26360	(load integral control gain of 26360)
	RSD7	(reset digital output 7, i.e. turn the "press running" output off)
	STD8	(set digital output 8, i.e. turn the "press home" output on)
	RSB1	(reset boolean variable 1)
	MED	(end motion block 2 and exit motion block editor)

6. Drilling Example

This example is an extended version of the Multiple Speed Position Moves example (example 3). It completes example 3 by adding commands that setup the IMC system, a homing routine, and a fault handling routine.

```
                CC=100          (load continuous current of 100% of continuous current rating of
                                 drive)
                PC=100          (load peak current of 100% of peak current rating of drive)
                KP=80           (load proportional control gain of 80)
                KI=23310        (load integral control gain of 23310)
                KD=554          (load derivative control gain of 554)
                KA=170          (load acceleration feedforward of 170)
                FT=1            (load filter time constant of 1)
                FP=10           (assign floating point variable allocation of 10)
                AUR=4096        (load axis unit ratio of 4096 pulses/unit, therefore units=revolutions)
                FB=0.1          (load following error bound of 0.1 units)
                PB=0            (load in-position band of 0 units)
                PWE=0           (disable position register wrap)
                MD=CW           (assign direction of motor for forward moves of "clockwise")

Program 1       DE1             (define program 1)
(main         1 IF EG1 AND N PX2
program)            GS10        (conditionally gosub 10 if positive edge sensitive digital input 1, i.e.
                                 the "run drill" input, is active and program 2 is not executing)
                IF EG2 GS20     (conditionally gosub 20 if positive edge sensitive digital input 2, i.e.
                                 the "stop drill" input, is active)
                IF EG3 AND N PX2
                    GS30        (conditionally gosub 30 if positive edge sensitive digital input 3, i.e.
                                 the "home drill" input, is active and program 2 is not executing)
                IF EG4 AND N PX2
                    GS40        (conditionally gosub 40 if positive edge sensitive digital input 4, i.e.
                                 the "find home" input, is active and program 2 is not executing)
                GT1             (unconditionally goto 1)
               10 IF MB OR N BV1 GT15 (conditionally goto 15 if a motion block is executing or boolean
                                 variable 1 is reset, i.e. the "home position found" boolean variable is
                                 set to 0)
                EX2             (execute program 2, which is the "run & monitor drill" program)
               15 RET           (return from gosub)
               20 KL2           (kill program 2)
                DC=200          (load deceleration of 200 units/sec^2)
                ST              (stop all motion)
                MEX2            (execute motion block 2, which is the "stop drill" motion block)
                RET             (return from gosub)
               30 IF MB OR N BV1 GT35 (conditionally goto 35 if a motion block is executing or boolean
                                 variable 1 is reset, i.e. the "home position found" boolean variable is
                                 set to 0)
                MEX3            (execute motion block 3, which is the "home drill" motion block)
               35 RET           (return from gosub)
               40 IF MB GT 45   (conditionally goto 45 if a motion block is executing)
                EX3             (execute program 3, which is the "find home" program)
               45 RET           (return from gosub)
                ED              (end program 1 and exit program editor)
```

Program 2		DE2	(define program 2)
("run & monitor		MEX1	(execute motion block 1, which is the "run drill" motion block)
drill")		WT DO9	(wait for digital output 9 to be set, i.e. wait for the "coolant on" output to turn on)
	1	FV1=UC	(load floating point variable 1 with the control output)
		IF FV1>45 GT 5	(conditionally goto 5 if floating point variable 1 is greater than 45, i.e. if the control output is greater than 45% of maximum)
		IF N DO9 GT10	(conditionally goto 10 if digital output 9 is reset, i.e. if the "coolant on" output turns off)
		GT1	(unconditionally goto 1)
	5	STD10	(set digital output 10, i.e. turn the "dull drill bit" output on)
	10	ED	(end program 2 and exit program editor)
Program 3		DE3	(define program 3)
("find home")		IF N IO8 GT10	(conditionally goto 10 if I/O register bit 8 is set to 0, i.e. if the home input is not active)
		VL=1	(load velocity of 1 unit/sec)
		AC=50	(load acceleration of 50 units/sec^2)
		RFN	(run forward)
		WT N IO8	(wait for I/O register bit 8 to be set to 0, i.e. the home input to become inactive)
		ST	(stop all motion)
		WT IP	(wait for axis to be in position)
	10	HRH	(home reverse to home input)
		WT IP	(wait for axis to be in position)
		HFM	(home forward to marker)
		WT IP	(wait for axis to be in position)
		SAP:0	(set absolute position register equal to 0)
		STD8	(set digital output 8, i.e. the "drill home" output)
		STB1	(set boolean variable 1, i.e. set the "home position found" boolean variable to 1)
		ED	(end program 3 and exit program editor)
Motion Block 1		MB1	(define motion block 1)
("run drill")		RSD8	(reset digital output 8, i.e. turn the "drill home" output off)
		STD7	(set digital output 7, i.e. turn the "drill operating" output on)
		VL=20	(load velocity of 20 units/sec)
		AC=200	(load acceleration of 200 units/sec^2)
		AM=15	(load absolute move position of 15 units)
		RAN	(run to absolute move position)
		VL=5	(load velocity of 5 units/sec)
		STD9	(set digital output 9, i.e. turn the "coolant on" output on)
		AC=40	(load acceleration of 40 units/sec^2)
		AM=20	(load absolute move position of 20 units)
		RAN	(run to absolute move position)
		RSD9	(reset digital output 9, i.e. turn the "coolant on" output off)
		VL=20	(load velocity of 20 units/sec)
		AC=200	(load acceleration of 200 units/sec^2)
		AM=0	(load absolute move position of 0 units)
		RAN	(run to absolute move position)
		RSD7	(reset digital output 7, turn the "drill operating" output off)
		STD8	(set digital output 8, turn the "drill home" output on)
		MED	(end motion block 1 and exit motion block editor)

Motion Block 2 ("stop drill")	MB2	(define motion block 2)
	WT IP	(wait for axis to be in position)
	RSD9	(reset digital output 9, i.e. turn the "coolant on" output off)
	RSD7	(reset digital output 7, i.e. turn the "drill operating" output off)
	MED	(end motion block 2 and exit motion block editor)
Motion Block 3 ("home drill")	MB3	(define motion block 3)
	VL=20	(load velocity of 20 units/sec)
	AC=200	(load acceleration of 200 units/sec^2)
	AM=0	(load absolute move position of 0 units)
	RAN	(run to absolute move position)
	RSD7	(reset digital output 7, turn the "drill operating" output off)
	STD8	(set digital output 8, turn the "drill home" output on)
	MED	(end motion block 3 and exit motion block editor)
Program 4 (fault handling routine)	DE4	(define program 4)
	RSD7	(reset digital output 7, i.e. turn the "drill operating" output off)
	RSD8	(reset digital output 8, i.e. turn the "drill home" output off)
	RSD9	(reset digital output 9, i.e. turn the "coolant on" output off)
	IV1=FC AND 1	(load integer variable 1 with the result of FC AND 1, i.e. the value of fault code status register bit 0; fault code status register bit 0 is set to 1 after a power failure has occurred)
	IF IV1<>0 GS10	(conditionally gosub 10 if integer variable 1 is not equal to 0)
	IV1=FC AND -10	(load integer variable 1 with the result of FC AND -10, i.e. the integer value of FC AND -10, which not will be equal to 0 if a fault code status register bit other than bit 3 or bit 0 is set [note that -10_{10} = $11111111111111111111111111110110_2$]; i.e., FC AND -10 will not be equal to 0 if some fault other than a power failure or lost enable has occurred)
	IF IV1<>0 GT20	(conditionally goto 20 if integer variable 1 is not equal to 0)
	GT30	(unconditionally goto 30)
10	DOE=1	(enable digital outputs)
	RSB1	(reset boolean variable 1, i.e. set the "home position found" boolean variable to 0)
	RET	(return from gosub)
20	WT EG6	(wait for positive edge sensitive digital input 6, i.e. the "reset faults" input, to be active)
30	WT IOB	(wait for I/O register bit 11 to be set to 1, i.e. for the enable input to be active)
	RSF	(reset faults)
	RSD10	(reset digital output 10, i.e. turn "dull drill bit" output off)
	IF N BV1 GT40	(conditionally goto 40 if boolean variable 1 is reset, i.e. if the "home position found" boolean variable is set to 0)
	FV1=AP	(load floating point variable 1 with the axis position)
	IF FV1<>0 GT40	(conditionally goto 40 if floating point variable 1 is not equal to 0)
	STD8	(set digital output 8, i.e. turn the "drill home" output on)
40	EX1	(execute program 1, which is the main program)
	ED	(end program 4 and exit program editor)

7. Electronic Gearing Example

This example uses the Enhanced Version's auxiliary encoder input and analog input to implement variable ratio electronic gearing. The motor controlled by the IMC-313E-X-C is synchronized to the auxiliary encoder input so that the motor runs at a ratio to the encoder. The ratio is controlled by the voltage at the analog input. This creates a variable ratio transmission where the ratio can be set by an analog voltage value.

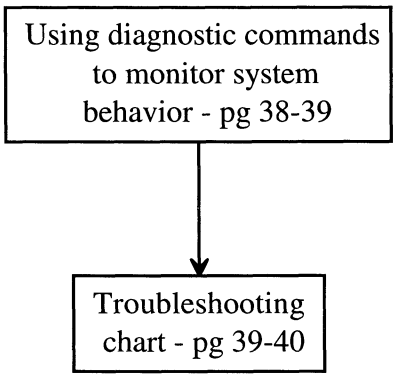
Note that for every 2 V on the analog input, the encoder ratio changes by one. In other words, for AI = 2 V, RN/RD = 1; for AI = 4 V, RN/RD = 2; and so on up to AI = 10 V, RN/RD = 5.

```
( SYSTEM COMPONENTS

( Controller:  IMC-316E-X-C
( Motor:    Whedco MTR series with single speed resolver feedback
( Encoder:  1000 line quadrature with differential line driver outputs
( Analog input:   0 - 10 vdc

( DIGITAL I/O ASSIGNMENTS
( DI1:  "run" input

( VARIABLE ASSIGNMENTS

( IV1:  general purpose integer
( FV1:  calculated encoder position
( FV2:  difference between encoder position and FV1
( FV3:  step size for stepping up gearing numerator
( FV4:  calculated gearing numerator
( FV10: value of analog input/2 * 1000
( FV11: general purpose floating point
( FV12: general purpose floating point
( FV13: general purpose floating point
( BV1:  boolean toggle for gearing enabled flag

( PARAMETER INITIALIZATION

AUR=4096            ( load axis unit ratio of 4096 counts per rev )
XUR=1000            ( load encoder unit ration of 1000 counts per rev )
XI=Q1               ( set encoder to quadrature one )
PWE=1               ( enable position register wrap )
XPL=1000            ( load auxiliary position length )
RN=0                ( load electronic gearing numerator )
RD=1000             ( load electronic gearing denominator )
RIE=1               ( enable the gearing, i.e. ratio input )
DFT=0.05            ( set digital input filter time )
DIF=000000000001    ( assign filter to digital input 1 )
CC=100              ( load continuous current )
FB=0.1              ( load following error bound )
KP=80               ( load proportional control gain )
KI=23310            ( load integral control gain )
KD=300              ( load derivative control gain )
KA=0                ( load acceleration feedforward )
FT=1                ( load filter time constant )
FP=500              ( assign floating point variable allocation of 500 )
```

```
( PROGRAM 1 is the main program )
      DE1                    ( define program 1 )
   001WT DI1                 ( wait for "run" input to turn on )
      FV10=AI*500            ( load FV10 with the analog input times 500 )
      GS100                  ( gosub 100, which is the start axis routine )
      WT N DI1               ( wait for "run" input to turn off )
      FV10=RN                ( load FV10 with the gearing numerator )
      GS200                  ( gosub 200, which is the stop axis routine )
      GT1                    ( goto 1 )

      ( start axis routine )
   100FV1=XP                 ( load FV1 with encoder position )
      FV3=FV10/11            ( load FV3 with gearing numerator step size )
      FV11=FV10 - FV3        ( load FV11 with bound on calculated gearing numerator )
      FV4=FV3                ( load FV4 with FV3 )
      RN=FV4                 ( load gearing numerator with FV4 )
   ( ramp up gearing numerator RN from 0 to value from analog input )
   101FV1=FV1 + 0.1          ( increment calculated position by 0.1 )
      IF FV1 < 999.9 GT102   ( goto 102 if FV1 is less than 999.9 )
      FV1=FV1-2000           ( wrap around calculated value )
   102FV2=XP - FV1           ( load FV2 with encoder position minus calculated value )
      IF FV2 < 0 GT102       ( goto 102 if FV2 is less than 0 )
      IF FV2 > 1 GT102       ( goto 102 if FV2 is greater than 1 )
      FV4=FV4 + FV3          ( increment FV4 by gearing numerator step size )
      IF FV4 > FV11 GT103    ( goto 103 if FV4 is greater than numerator bound )
      RN=FV4                 ( load gearing numerator with FV4 )
      GT101                  ( end of ramp up loop )
   103RN=FV10                ( load gearing numerator with value from analog input )
      STB1                   ( set "gearing enabled" flag )
      RET                    ( return from gosub )

   ( stop axis routine )
   200RSB1                   ( reset "gearing enabled" flag )
      FV1=XP                 ( load FV1 with encoder position )
      FV3=FV10/11            ( load FV3 with gearing numerator step size )
      FV11=FV3               ( load FV11 with gearing numerator bound )
      FV4=FV10 - FV3         ( load FV4 with current numerator minus step size )
      RN=FV4                 ( load gearing numerator with FV4 )
   ( ramp down gearing numerator from current value to 0 )
   201FV1=FV1 + 0.1          ( increment calculated position by 0.1 )
      IF FV1 < 999.9 GT202   ( goto 202 if FV1 is less than 999.9 )
      FV1=FV1 - 2000         ( wrap around calculated value )
   202FV2=XP - FV1           ( load FV2 with encoder position minus calculated position )
      IF FV2 < 0 GT202       ( goto 202 if FV2 is less than 0 )
      IF FV2 > 1 GT202       ( goto 202 if FV2 is greater than 1 )
      FV4=FV4 - FV3          ( decrement FV4 by gearing numerator step size )
      IF FV4 < FV11 GT203    ( goto 103 if FV4 is less than numerator bound )
      RN=FV4                 ( load gearing numerator with FV4 )
      GT201                  ( end of ramp down loop )
   203RN=0                   ( load gearing numerator with 0 )
      RET                    ( return from gosub )
      ED                     ( end of program 1 )
```

```
( PROGRAM 2 sets the gearing ratio based on the analog input while gearing is enabled )
   DE2                  ( define program 2 )
001IF N BV1 GT1         ( goto 1 if "gearing enabled" flag reset )
   FV11=AI*500          ( load FV11 with the analog input divided by 2 times 1000)
   FV12=RN              ( load FV12 with the gearing numerator )
   FV13=FV11 - FV12     ( load FV13 with FV11 minus FV12 )
   FV13=ABS FV13        ( take absolute value )
   IF FV13 < .5 GT2     ( goto 2 if difference between AI & RN not significant )
   RN=FV11              ( load gearing numerator with FV11 )
002GT1                  ( end of loop )
   ED                   ( end program 2 )

( PROGRAM 4 Executes on Power Cycle or System Fault )
   DE4                  ( define program 4 )
   KL1                  ( kill program 1 )
   KL2                  ( kill program 2 )
   WT N IOB             ( wait for enable input to be inactive )
   STM1=.1              ( wait 0.1 seconds )
   WT TM1
   WT IOB               ( wait for enable input to be active )
   STM1=.1              ( wait 0.1 seconds )
   WT TM1
   RSB1                 ( reset "gearing enabled" flag )
   RSF                  ( reset fault )
   EX1                  ( execute main program )
   EX2                  ( execute "set gearing ratio" program )
   ED                   ( end program 4 )
```

3. Diagnostics/Troubleshooting



3. Diagnostics/Troubleshooting

Now that you have your programs/motion blocks up and running, it is somewhat reasonable to think that everything will work as expected. Unfortunately, it is always possible that something may go wrong. If this happens, you will want to correct the error as quickly as possible so that system down time can be minimized. The purpose of this chapter is to show you how to effectively determine and correct system problems. This can be done in two ways:

A. using the diagnostic commands to monitor system behavior and
B. going through a troubleshooting process when problems occur.

A. Diagnostic Commands

This section explains the use of diagnostic commands to determine system problems. First, the diagnostic command summary is given. Then, the following topics will be discussed:

1. how to use diagnostic items and conditions and
2. how to use the single step mode to debug programs.

Diagnostic Command Summary

mnemonic	*description*
REV	reports software revision
RMR	reports memory remaining
RBV	reports boolean variable value
RIV	reports integer variable value
RFV	reports floating point variable value
RSV	reports string variable value
RDO	reports digital output
RDI	reports digital input
REG	reports positive edge sensitive digital input
RAO	reports analog output
RTC	reports real time clock value
RTM	reports timer value
RPS	reports program status
RSRS	reports system status register
RSRA	reports axis status register
RIO	reports I/O register
RFI	reports fault input register
RFC	reports fault code
RCP	reports command position
RAP	reports axis position
RAT	reports axis position capture
RAV	reports axis velocity
RUC	reports control output
RFE	reports following error
RRR	reports resolver reading
DGE	enables diagnostics
DGL	prints diagnostic line of items
DGP	prints diagnostic message to terminal
DGS	sets program to single step mode
DGC	assigns diagnostic condition for printing
DGI	assigns diagnostic item to be printed

1. Diagnostic Conditions and Items (DGC, DGI; DGE, DGL)

Diagnostic conditions and items are used to help distinguish certain events in the system that may be causing problems. Diagnostic items are simply expression operands that are used in the context of the diagnostic mode, which is enabled by the DGE command. Up to eight diagnostic items can be defined using the DGI command. Diagnostic items, for example, can be things like the axis position (AP), axis velocity (AV), control output (UC), or variable values (BV*n*, IV*n*, etc.). A line of diagnostic items can be printed to the terminal using the DGL command. See the command summary for more information.

Alternatively, you can tell the controller when to print a diagnostic line of items to the terminal by setting diagnostic conditions. When a diagnostic condition is met, then a diagnostic line items will be printed to the terminal. The DGC command can be used to define up to four diagnostic conditions. Diagnostic conditions can be any boolean expression, but specifically can be program *n* executing (PX*n*), timer *n* timed out (TM*n*) or motion generator enabled (SRA0). Note that only one of the four diagnostic conditions needs to be satisfied for the controller to output the diagnostic line of items.

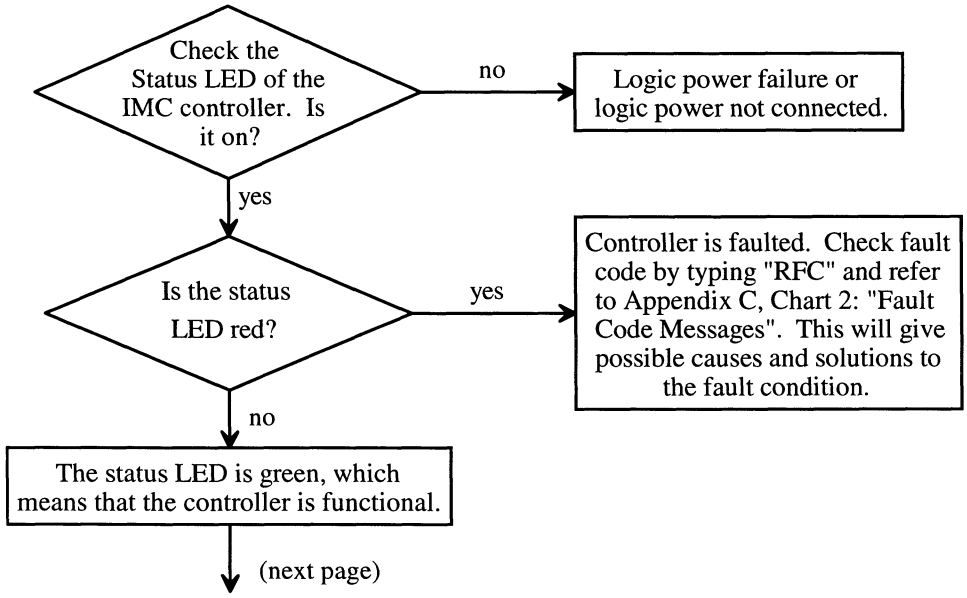
For more information on boolean expressions (bexpr), See Appendix B, "Expression Charts".

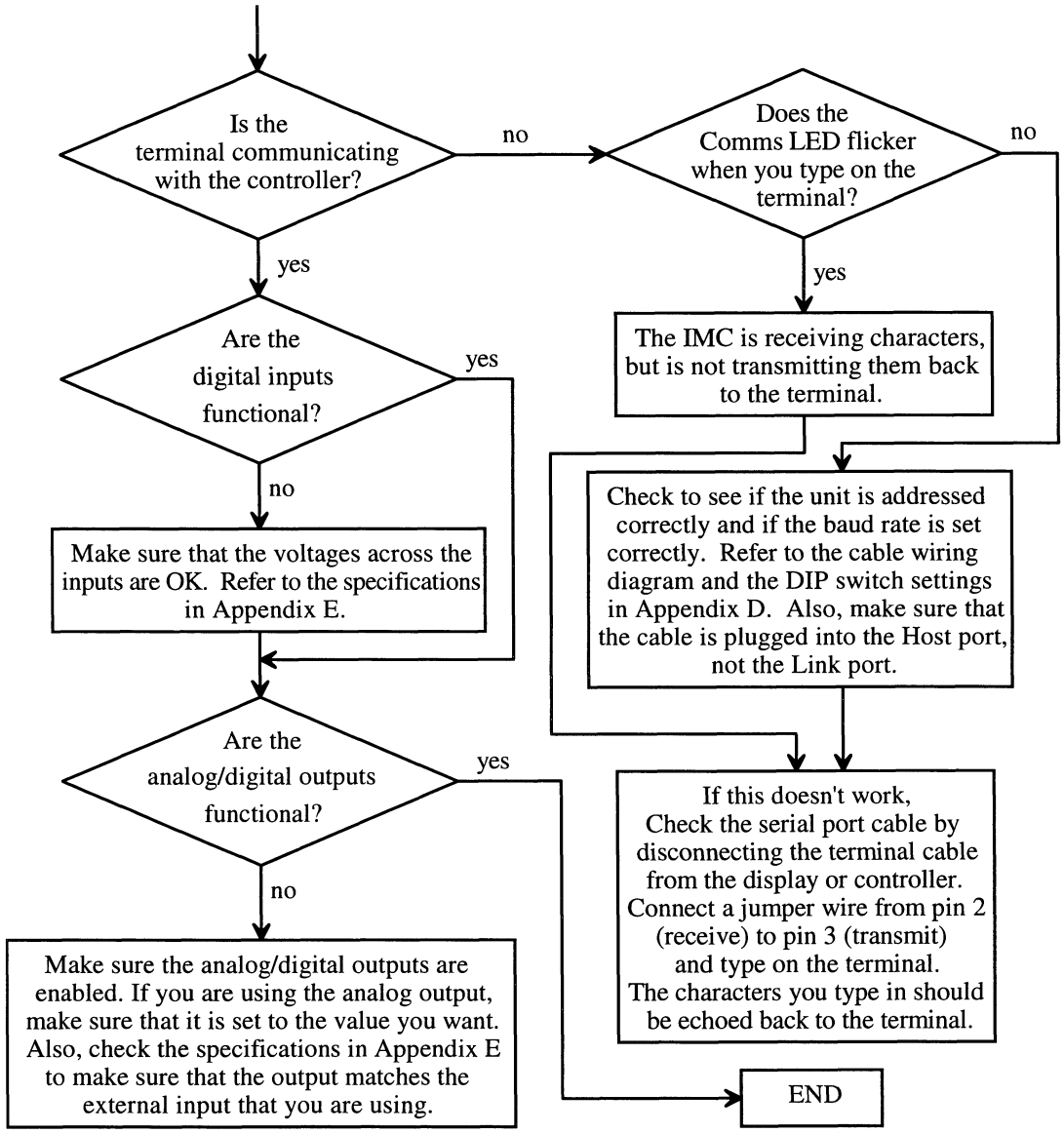
2. Single Step Mode (DGS)

The single step mode of the controller is used to diagnose problems that occur as a result of program errors. Once it is enabled, you can single step through a program using the X command. The single step mode is enabled by setting DGS to the number of the program you want to single step through. To single step through a program means executing a program one line at a time. So, when you issue the X command in single step mode, only one line of the program will be executed. This action can be repeated until you have reached the last statement of the program. In addition, if a number *n* is used with the X command (X *n*), then you will step through *n* lines of the program. For more information, see Appendix A, "Command Summary".

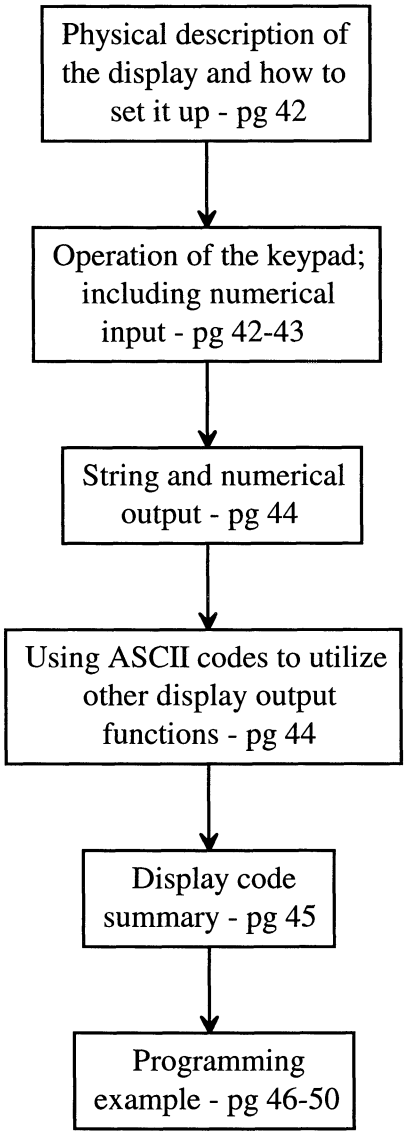
B. Troubleshooting

The purpose of this section is to help you to determine and correct problems that may occur with your IMC system by giving a troubleshooting chart to follow. If the problem is still unresolved after following these procedures, contact Whedco customer service for assistance.





4. Using The Display



4. Using The Display

This chapter describes how to use the display, which may be utilized as an operator interface for the controller system. For example, it can be used to display and/or change system parameters, such as cutting length or drilling depth, whenever needed. It can also be used to perform other functions, such as axis jogging or homing. The following topics will be covered in this chapter:

A. The physical description of the display;
B. operation of the keypad;
C. how to use the output capabilities of the display; and
D. programming examples of how the display is used.

A. Physical Description

The display is made up of two parts; first, the LCD itself, and second, the keypad. The LCD provides the means for textual output. It can display a maximum of 4 lines of output, each of which can be up to 40 characters long. The keypad is divided into two parts: the left hand side, which is the alpha keypad, and the right hand side, which is the numerical keypad. Pressing a key on the alpha keypad will send one of the ASCII characters "A" through "L" to the serial port. The numerical keypad is used to send numbers to the serial port.

The main use of the alpha keypad is to implement controller functions, for example, "start", "stop", "home axis", "jog forward", etc. For this reason, a card that shows what function is represented by what key can be inserted above the alpha keypad. There is one thin mylar card included in the display to be used for this purpose. To remove the card, you should unscrew the four screws on the front of the display, take the front bezel off, and then manually remove the card from the pocket. Make sure to use Section B, "Keypad Operation", as a reference before writing to or typing on the card. After doing this, you can then slide the card back into the pocket and screw the front bezel back on to the display.

The display should be placed in between the terminal and the controller for operation. In other words, the terminal should be serially connected to the display, which should be serially connected to the controller. For details on the display connections, see Appendix D.

B. Keypad Operation

As stated above, the alpha keypad is used to send one of the ASCII characters "A" through "L" to the serial port, and the numerical keypad is used to send numbers to the serial port. Also, note that both pressing and releasing a key on the alpha keypad results in a character being sent to the key buffer. This feature can be useful in operations that only need to be executed for a short period of time by holding down a key, such as axis jogging. A list of the description of the keys on the keypad follows. The description of the keys on the alpha keypad includes a set of parentheses with two numbers separated by a comma. These numbers tell where the key is located. The first number represents the row of the alpha keypad, and the second number represents the column of the alpha keypad.

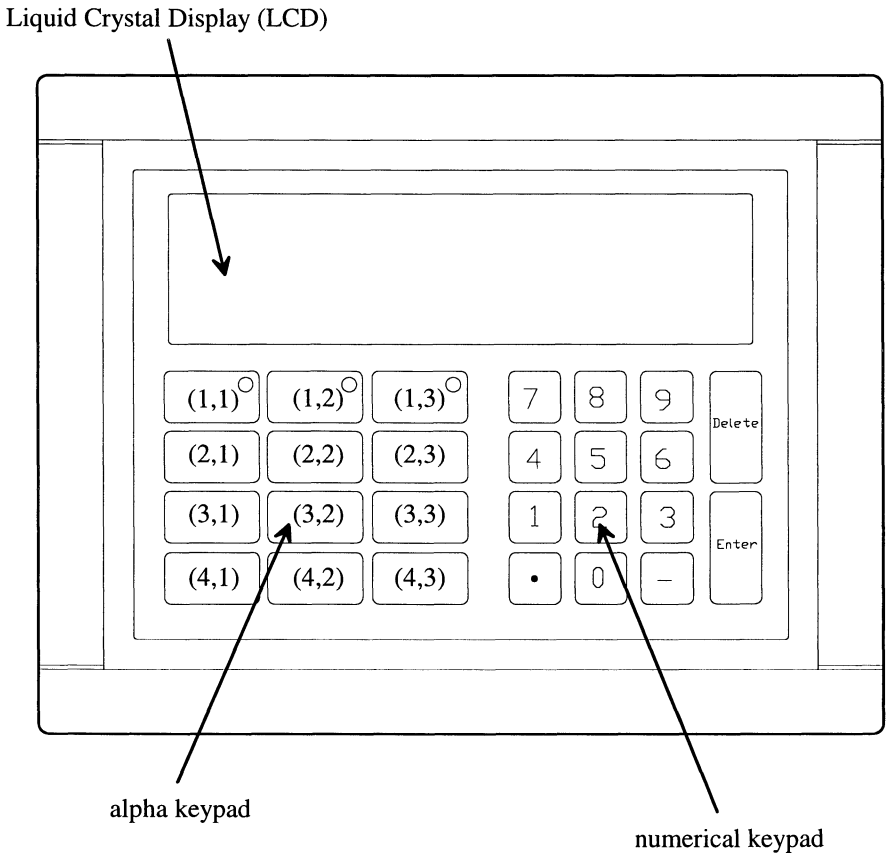
Also, a drawing of the front panel of the display is shown for reference.

Description	Character sent when pressed	Character sent when released
Alpha key (1,1)	"A" (ASCII 65)	"a" (ASCII 97)
Alpha key (1,2)	"B" (ASCII 66)	"b" (ASCII 98)
Alpha key (1,3)	"C" (ASCII 67)	"c" (ASCII 99)
Alpha key (2,1)	"D" (ASCII 68)	"d" (ASCII 100)
Alpha key (2,2)	"E" (ASCII 69)	"e" (ASCII 101)
Alpha key (2,3)	"F" (ASCII 70)	"f" (ASCII 102)

Description	Character sent when pressed	Character sent when released
Alpha key (3,1)	"G" (ASCII 71)	"g" (ASCII 103)
Alpha key (3,2)	"H" (ASCII 72)	"h" (ASCII 104)
Alpha key (3,3)	"I" (ASCII 73)	"i" (ASCII 105)
Alpha key (4,1)	"J" (ASCII 74)	"j" (ASCII 106)
Alpha key (4,2)	"K" (ASCII 75)	"k" (ASCII 107)
Alpha key (4,3)	"L" (ASCII 76)	"l" (ASCII 108)

Description	Character sent when pressed	Character sent when released
0	"0" (ASCII 48)	none
1	"1" (ASCII 49)	none
2	"2" (ASCII 50)	none
3	"3" (ASCII 51)	none
4	"4" (ASCII 52)	none
5	"5" (ASCII 53)	none
6	"6" (ASCII 54)	none

Description	Character sent when pressed	Character sent when released
7	"7" (ASCII 55)	none
8	"8" (ASCII 56)	none
9	"9" (ASCII 57)	none
"."	"." (ASCII 46)	none
"-"	"-" (ASCII 45)	none
Delete	del (ASCII 127)	none
Enter	cr (ASCII 13)	none



If, when using the IN command to input numbers, an incorrect number is input, one of the program status register bits will be set. Bit 4 will be set if the number has an invalid digit, and bit 5 will be set if the number is out of the range of the variable used, whether it be a floating point or integer variable. See "Program Status Messages" (Appendix C, Chart 5) for more information.

Also, when using the display to input numbers, it is helpful to disable the alpha keypad so that no other characters can be accidentally input. This can be done by sending the controller two ASCII codes. Specifically, issue "ASC27" and then "ASC60", which is the ASCII code sequence that tells the display to disable the alpha keypad. To enable the alpha keypad, issue "ASC27" followed by "ASC62". For more about using ASCII codes, see section C, "Display Output".

C. Display Output

To send output to the display, you can use (1) PRT and PNT to print strings on the display, (2) the OUT command to send numbers to the display or (3) the ASC command to utilize other display output functions.

1. String Output (PRT, PNT)

Character strings are output to the display using the PRT and PNT commands. The PRT command prints a string to the display with a carriage return and a line feed. The PNT command prints a string to the display without the carriage return and line feed. Printing a string in this way can be useful if you wish to print a variable value as part of a single line of characters. For example, if you wanted to print "Axis velocity =" followed by the numerical value of the axis velocity, use the command sequence "PNT"Axis velocity =", FV1=AV, OUT FV1". This can be followed by the command sequence "ASC13, ASC10" if you want the cursor to advance to the next line.

2. Numerical Output (OUT, FW, DP)

Variable values are output to the display using the OUT command. The number is displayed with a certain format which is set by the controller and the FW (field width) and DP (decimal places) parameters. The field width is the number of characters that the controller will use to display the number, and the decimal places parameter is the number of decimal places that will be displayed as part of the field. If a number will not fit in the specified field width, then the controller prints asterisks (*) in the field. Numbers are always right justified in the field and all specified decimal places are printed, even if they are zero.

When OUT is used, integer variables are always printed in the integer format and floating point variables are always printed in the floating point format. The exponent form is never used for integer variables. When E is printed, it always takes up three places, for example, E12 or E01. Minus exponents will never be printed since numbers less than the last displayed decimal place in magnitude are displayed as zero.

3. Using ASCII Codes (ASC)

In order to make use of the display outside of sending data to it using the PRT, PNT and OUT commands, you must use the ASC command to send ASCII codes. There are three ways in which you can use ASCII codes:

a. to format data on the display,
b. to turn the three LEDs on and off, or
c. to disable and enable the alpha keypad.

Data formatting can be done by sending general ASCII codes. For example, "ASC13" sends a carriage return to the display. A list of general ASCII codes is shown on the next page. All other functions including data formatting can be accomplished using escape codes, which are ASCII codes that are sent to the display after the escape character has been sent. The escape character is defined by ASCII code 27. So, to send an escape code to the display, you must issue "ASC27", and then issue the ASC command again with the escape code you wish to send. For example, "ASC27" followed by "ASC74" clears the display and homes the cursor. A list of escape codes is also shown on the next page.

Also, positioning the cursor is accomplished by first sending "ASC27" and then "ASC71" to the display, after which two more ASCII codes are sent. These define the horizontal and vertical position of the cursor. For example, sending "ASC27", "ASC71", "ASC17", "ASC2" will position the cursor in the 17th space of the 2nd line of the LCD.

The LEDs, which are located in the upper right hand corner of the top three alpha keys, can be turned on and off using escape codes, as shown in the above table. The LEDs can be used in conjunction with the alpha key it is located in. For example, it can be used to show if the function defined by the alpha key is currently active.

Disabling and enabling the alpha keypad is accomplished by sending escape codes to the display, as mentioned in Section B, "Keypad Operation". This can be useful when entering the numerical values of system parameters.

Display Code Summary

1. General ASCII Codes

Code	Description	Usage
8	backspace	This code moves the cursor back one space and prints a space.
10	line feed	This code moves the cursor down one line.
13	carriage return	This code moves the cursor to the leftmost space.
27	escape	This code lets the display know that the next ASCII code which will be sent is an escape code.

2. Escape Codes

Code	Description	Usage
49	LED1 on	This code turns LED1 on.
50	LED2 on	This code turns LED2 on.
51	LED3 on	This code turns LED3 on.
52	LED1 off	This code turns LED1 off.
53	LED2 off	This code turns LED2 off.
54	LED3 off	This code turns LED3 off.
60	alpha off	This code disables the alpha keypad.
62	alpha on	This code enables the alpha keypad.
65	cursor up	This code moves the cursor up one line.
66	cursor down	This code moves the cursor down one line.
67	cursor right	This code moves the cursor right one space.
68	cursor left	This code moves the cursor left one space.
70	cursor position	This code places the cursor in a specific position defined by the next two ASCII codes sent. The first is the horizontal position with an offset of 32 (33-62) and the second is the vertical position with an offset of 32 (33-36).
72	cursor home	This code homes the cursor, i.e. moves it to the upper left hand corner of the screen.
73	clear line	This code clears the current line and places the cursor at the beginning of the line.
74	clear display	This code clears the display and homes the cursor.

D. Programming Example

The following is an example of how the display can be used in a program. The program itself is a basic demonstration of the capabilities of the controller and the display. The following functions are defined using the alpha keypad:

"A"	Start demo	"G"	Enter new velocity
"E"	Start axis motion	"H"	Enter new distance
"F"	End axis motion	"J"	Jog axis forward
		"K"	Jog axis reverse

Note that jogging cannot be done while the axis is moving.

```
                    FP=10               (assign floating point variable allocation of 10)
                    AUR=4096            (load axis unit ratio of 4096 pulses/unit, therefore units=revolutions)
                    AM=0                (load absolute move position of 0 units)
                    IM=10               (load incremental move position of 10 units)
                    VL=5                (load velocity of 5 units/sec)
                    AC=50               (load acceleration of 50 units/sec^2)
                    FW=9                (assign field width of output variable on display)
                    DP=3                (assign number of places after decimal point on display)
                    DSE=1               (enable display format on serial port)

Program 1           DE1                 (define program 1)
(main program)      STM1=.25            (load start time of timer 1 and start timer 1)
                1   IF N KY GT1         (conditionally goto 1 if no character in key buffer)
                    GETIV1              (get one character from key buffer)
                    IF IV1<>65 GT1      (conditionally goto 1 if integer variable 1 not equal to 65, i.e. ASCII
                                         "A")
                    STM2=.2             (load start time of timer 2 and start timer 2)
                    WTTM2               (wait for timer 2 to be timed out)
                    ASC27               (send ASCII code 27, i.e. the "escape" character, to the display)
                    ASC49               (send ASCII code 49 to the display, i.e. turn LED1 of the display on)
                    ASC27               (send ASCII code 27, i.e. the "escape" character, to the display)
                    ASC74               (send ASCII code 74 to the display, i.e. clear the display and home the
                                         cursor)
(check for      10  IF TM1 GS20         (conditionally gosub 20 if timer 1 is timed out)
alpha key           IF N KY GT10        (conditionally goto 10 if no character in key buffer)
input)              GETIV1              (get one character from key buffer)
                    IF IV1=69 GT30      (conditionally goto 30 if integer variable 1 equal to 69, i.e. ASCII "E")
                    IF IV1=70 GT40      (conditionally goto 40 if integer variable 1 equal to 70, i.e. ASCII "F")
                    IF IV1=71 GT50      (conditionally goto 50 if integer variable 1 equal to 71, i.e. ASCII "G")
                    IF IV1=72 GT60      (conditionally goto 60 if integer variable 1 equal to 72, i.e. ASCII "H")
                    IF IV1=74 GT70      (conditionally goto 70 if integer variable 1 equal to 74, i.e. ASCII "J")
                    IF IV1=75 GT80      (conditionally goto 80 if integer variable 1 equal to 75, i.e. ASCII "K")
                    GT10                (unconditionally goto 10)
(update         20  STM1=.25            (load start time of timer 1 and start timer 1)
clock, axis         ASC27               (send ASCII code 27, i.e. the "escape" character, to the display)
velocity &          ASC72               (send ASCII code 72 to the display, i.e. home the cursor)
position)           SV1=RTC             (load string variable 1 with the real time clock value)
                    OUTSV1              (output string variable 1 value to the display)
                    ASC13               (send ASCII code 13, i.e. a carriage return, to the display)
                    ASC10               (send ASCII code 10, i.e. a line feed, to the display)
                    PNT"Servo Velocity: "  (print string to display without carriage return and line feed)
```

```
                FV1=AV                  (load floating point variable 1 with the axis velocity)
                OUTFV1                  (output floating point variable 1 value to the display)
                PNT" rev/sec"           (print string to display without carriage return and line feed)
                ASC13                   (send ASCII code 13, i.e. a carriage return, to the display)
                ASC10                   (send ASCII code 10, i.e. a line feed, to the display)
                PNT"Servo Position: "   (print string to display without carriage return and line feed)
                FV1=AP                  (load floating point variable 1 with the axis position)
                OUTFV1                  (output floating point variable 1 value to the display)
                PNT" rev"               (print string to display without carriage return and line feed)
                ASC13                   (send ASCII code 13, i.e. a carriage return, to the display)
                ASC10                   (send ASCII code 10, i.e. a line feed, to the display)
                RET                     (return from gosub)
(start axis  30 STB1                    (set boolean variable 1, i.e. set it to 1)
motion)         EX2                     (execute program 2)
                GT10                    (unconditionally goto 10)
(end axis    40 RSB1                    (reset boolean variable 1, i.e. set it to 0)
motion)         GT10                    (unconditionally goto 10)
(enter new   50 ASC27                   (send ASCII code 27, i.e. the "escape" character, to the display)
velocity)       ASC60                   (send ASCII code 60 to the display, i.e. disable the alpha keypad)
                GETIV1                  (get one character from key buffer)
             52 PNT"Enter new velocity:" (print string to display without carriage return and line feed)
                INFV1                   (input floating point 1 variable value from key buffer)
                IV1=PS1 AND 16          (load integer variable 1 with the result of "PS1 AND 16")
                IF IV1<>0 GT 54(conditionally goto 54 if integer variable 1 is not equal to 0, i.e.,
                                         program 1 status register bit 4 is set to 1)
                IV1=PS1 AND 32          (load integer variable 1 with the result of "PS1 AND 32")
                IF IV1<>0 GT 55(conditionally goto 55 if integer variable 1 is not equal to 0, i.e.,
                                         program 1 status register bit 5 is set to 1)
                IF FV1 < 1 GT51         (conditionally goto 51 if floating point variable 1 less than 1)
                IF FV1 > 30 GT51        (conditionally goto 51 if floating point variable 1 greater than 30)
                VL=FV1                  (load velocity of "FV1" units/sec)
                ASC27                   (send ASCII code 27, i.e. the "escape" character, to the display)
                ASC62                   (send ASCII code 62 to the display, i.e. enable the alpha keypad)
                ASC27                   (send ASCII code 27, i.e. the "escape" character, to the display)
                ASC73                   (send ASCII code 73 to the display, i.e. clear line and put cursor at
                                         beginning of line)
                GT10                    (unconditionally goto 10)
             51 ASC27                   (send ASCII code 27, i.e. the "escape" character, to the display)
                ASC73                   (send ASCII code 73 to the display, i.e. clear line and put cursor at
                                         beginning of line)
                PNT"Invalid velocity; must be 1 to 30"
                                        (print string to display without carriage return and line feed)
                STM2=2                  (load start time of timer 2 and start timer 2)
                WTTM2                   (wait for timer 2 to be timed out)
                ASC27                   (send ASCII code 27, i.e. the "escape" character, to the display)
                ASC73                   (send ASCII code 73 to the display, i.e. clear line and put cursor at
                                         beginning of line)
                GT52                    (unconditionally goto 52)
             54 ASC27                   (send ASCII code 27, i.e. the "escape" character, to the display)
                ASC73                   (send ASCII code 73 to the display, i.e. clear line and put cursor at
                                         beginning of line)
                PNT"Number entered has  invalid digit"
                                        (print string to display without carriage return and line feed)
```

```
                STM2=2              (load start time of timer 2 and start timer 2)
                WTTM2               (wait for timer 2 to be timed out)
                ASC27               (send ASCII code 27, i.e. the "escape" character, to the display)
                ASC73               (send ASCII code 73 to the display, i.e. clear line and put cursor at
                                     beginning of line)
                GT52                (unconditionally goto 52)
           55   ASC27               (send ASCII code 27, i.e. the "escape" character, to the display)
                ASC73               (send ASCII code 73 to the display, i.e. clear line and put cursor at
                                     beginning of line)
                PNT"Number entered is out of range"
                                    (print string to display without carriage return and line feed)
                STM2=2              (load start time of timer 2 and start timer 2)
                WTTM2               (wait for timer 2 to be timed out)
                ASC27               (send ASCII code 27, i.e. the "escape" character, to the display)
                ASC73               (send ASCII code 73 to the display, i.e. clear line and put cursor at
                                     beginning of line)
                GT52                (unconditionally goto 52)
(enter new      60 ASC27            (send ASCII code 27, i.e. the "escape" character, to the display)
distance)       ASC60               (send ASCII code 60 to the display, i.e. disable the alpha keypad)
                GETIV1              (get one character from key buffer)
                62 PNT"Enter new distance:" (print string to display without carriage return and line feed)
                INFV1               (input floating point 1 variable value from key buffer)
                IV1=PS1 AND 16      (load integer variable 1 with the result of "PS1 AND 16")
                IF IV1<>0 GT 64(conditionally goto 64 if integer variable 1 is not equal to 0, i.e.,
                                     program 1 status register bit 4 is set to 1)
                IV1=PS1 AND 32      (load integer variable 1 with the result of "PS1 AND 32")
                IF IV1<>0 GT 65(conditionally goto 65 if integer variable 1 is not equal to 0, i.e.,
                                     program 1 status register bit 5 is set to 1)
                IF FV1 < 1 GT61     (conditionally goto 61 if floating point variable 1 less than 1)
                IF FV1 > 100 GT61   (conditionally goto 61 if floating point variable 1 greater than 100)
                IM=FV1              (load incremental move position of "FV1" units)
                ASC27               (send ASCII code 27, i.e. the "escape" character, to the display)
                ASC62               (send ASCII code 62 to the display, i.e. enable the alpha keypad)
                ASC27               (send ASCII code 27, i.e. the "escape" character, to the display)
                ASC73               (send ASCII code 73 to the display, i.e. clear line and put cursor at
                                     beginning of line)
                GT10                (unconditionally goto 10)
           61   ASC27               (send ASCII code 27, i.e. the "escape" character, to the display)
                ASC73               (send ASCII code 73 to the display, i.e. clear line and put cursor at
                                     beginning of line)
                PNT"Invalid distance; must be 1 to 100"
                                    (print string to display without carriage return and line feed)
                STM2=2              (load start time of timer 2 and start timer 2)
                WTTM2               (wait for timer 2 to be timed out)
                ASC27               (send ASCII code 27, i.e. the "escape" character, to the display)
                ASC73               (send ASCII code 73 to the display, i.e. clear line and put cursor at
                                     beginning of line)
                GT62                (unconditionally goto 62)
           64   ASC27               (send ASCII code 27, i.e. the "escape" character, to the display)
                ASC73               (send ASCII code 73 to the display, i.e. clear line and put cursor at
                                     beginning of line)
                PNT"Number entered has invalid digit"
                                    (print string to display without carriage return and line feed)
```

```
                STM2=2                  (load start time of timer 2 and start timer 2)
                WTTM2                   (wait for timer 2 to be timed out)
                ASC27                   (send ASCII code 27, i.e. the "escape" character, to the display)
                ASC73                   (send ASCII code 73 to the display, i.e. clear line and put cursor at
                                         beginning of line)
                GT62                    (unconditionally goto 62)
             65 ASC27                   (send ASCII code 27, i.e. the "escape" character, to the display)
                ASC73                   (send ASCII code 73 to the display, i.e. clear line and put cursor at
                                         beginning of line)
                PNT"Number entered is out of range"
                                        (print string to display without carriage return and line feed)
                STM2=2                  (load start time of timer 2 and start timer 2)
                WTTM2                   (wait for timer 2 to be timed out)
                ASC27                   (send ASCII code 27, i.e. the "escape" character, to the display)
                ASC73                   (send ASCII code 73 to the display, i.e. clear line and put cursor at
                                         beginning of line)
                GT62                    (unconditionally goto 62)
(jog axis    70 IF PX2 GT72             (conditionally goto 72 if program 2 is executing)
forward)        FV2=VL                  (load floating point variable 2 with the velocity)
                VL=1                    (load velocity of 1 unit/sec)
                RFN                     (run forward)
             71 IF TM1 GS20             (conditionally gosub 20 if timer 1 is timed out)
                IF N KY GT71            (conditionally goto 71 if no character in key buffer)
                GET IV1                 (get one character from key buffer)
                IF IV1<>106 GT71        (conditionally goto 71 if integer variable 1 not equal to 106, i.e. ASCII
                                         "j")
                ST                      (stop all motion)
                VL=FV2                  (load velocity of "FV2" units/sec)
                GT10                    (unconditionally goto 10)
             72 ASC27                   (send ASCII code 27, i.e. the "escape" character, to the display)
                ASC70                   (send ASCII code 70 to the display, i.e. position the cursor)
                ASC33                   (send ASCII code 33 to the display, i.e. set the horizontal position of
                                         the cursor to be 1)
                ASC36                   (send ASCII code 36 to the display, i.e. set the vertical position of the
                                         cursor to be 4)
                PNT"Jogging not allowed."
                                        (print string to display without carriage return and line feed)
             73 IF N KY GT73            (conditionally goto 73 if no character in key buffer)
                GETIV1                  (get one character from the key buffer)
                ASC27                   (send ASCII code 27, i.e. the "escape" character, to the display)
                ASC73                   (send ASCII code 73 to the display, i.e. clear line and put cursor at
                                         beginning of line)
                GT10                    (unconditionally goto 10)
(jog axis    80 IF PX2 GT72             (conditionally goto 72 if program 2 is executing)
reverse)        FV2=VL                  (load floating point variable 2 with the velocity)
                VL=1                    (load velocity of 1 unit/sec)
                RRN                     (run reverse)
             81 IF TM1 GS20             (conditionally gosub 20 if timer 1 is timed out)
                IF N KY GT81            (conditionally goto 81 if no character in key buffer)
                GET IV1                 (get one character from key buffer)
                IF IV1<>107 GT81        (conditionally goto 81 if integer variable 1 not equal to 107, i.e. ASCII
                                         "k")
                ST                      (stop all motion)
```

	VL=FV2	(load velocity of "FV2" units/sec)
	GT10	(conditionally goto 10)
	ED	(end program 1 and exit program editor)
Program 2	DE2	(define program 2)
("move axis") 1	RIN	(run to incremental move position)
	WTIP	(wait for axis to be in position)
	STM3=.5	(load start time of timer 3 and start timer 3)
	WTTM3	(wait for timer 3 to time out)
	RAN	(run to absolute move position)
	WTIP	(wait for axis to be in position)
	STM3=.5	(load start time of timer 3 and start timer 3)
	WTTM3	(wait for timer 3 to time out)
	IF BV1 GT1	(conditionally goto 1 if boolean variable 1 is set, i.e. is equal to 1)
	ED	(end program 2 and exit program editor)
Program 4	DE4	(define program 4)
(fault handling	KL1	(kill program 1)
routine)	KL2	(kill program 2)
	ASC27	(send ASCII code 27, i.e. the "escape" character, to the display)
	ASC74	(send ASCII code 74 to the display, i.e. clear the display and home the cursor)
	ASC27	(send ASCII code 27, i.e. the "escape" character, to the display)
	ASC52	(send ASCII code 52 to the display, i.e. turn LED1 of the display off)
	WT N IOB	(wait for I/O register bit 11 to be set to 0, i.e. for the enable input to be inactive)
	WT IOB	(wait for I/O register bit 11 to be set to 1, i.e. for the enable input to be active)
	EKB	(empty key buffer)
	RSF	(reset faults)
	EX1	(execute program 1)
	ED	(end program 4 and exit program editor)

Appendix A

Command Summary

Command Summary Template

The text below shows a template of a page of the command summary. Note that not all of the fields (e.g., "Type", "Syntax") are used for all of the commands. Only the appropriate ones appear when necessary. In addition, the command syntax is displayed in more detail at the bottom of this page.

Command mnemonic — Command description

DGC	**ASSIGNS DIAGNOSTIC CONDITION FOR PRINTING**	
Type:	diagnostic	
Syntax:	DGC*p1*=*p2*	
Parameters:	*p1*	*p2*
type	integer	bexpr
default		NULL
minimum	1	
maximum	4	
allowed values		any boolean expression, NULL
Usage:	This command assigns diagnostic condition *p1*. When one of the user defined	
	diagnostic conditions is satisfied, and if diagnostics are enabled, a diagnostic line of items is sent to the terminal (see DGL, DGI).	
Restrictions:	not allowed in programs or motion blocks	
Remarks:	Upon clearing the memory with the CLM command, all diagnostic conditions and items are set to the value "NULL", which means that there are no diagnostic conditions/items assigned. If you wish to eliminate the assignment of diagnostic condition *p1*, use the DGC command and set parameter *p2* to "NULL". For example, DGC1=NULL will eliminate the assignment of diagnostic condition 1.	
Example:		
STM2=0.5	(load start time of timer 2, start timer 2)	
DGC1=TM2 AND PX1	(assign diagnostic condition 1)	
	What will happen:	
	By loading timer 2 and assigning diagnostic condition 1, when diagnostics are	
	enabled, a diagnostic line of items will be sent to the terminal every 0.5 seconds	
	while program 1 is executing. Each diagnostic line will begin with the diagnostic condition satisfied, which in this case would be "TM2 AND PX1", and then be followed by a colon and the diagnostic items loaded.	

Type of command (i.e., system, axis, program, etc.)

Command Syntax

Command Parameters

(note that parameters may also have "units" assigned to them)

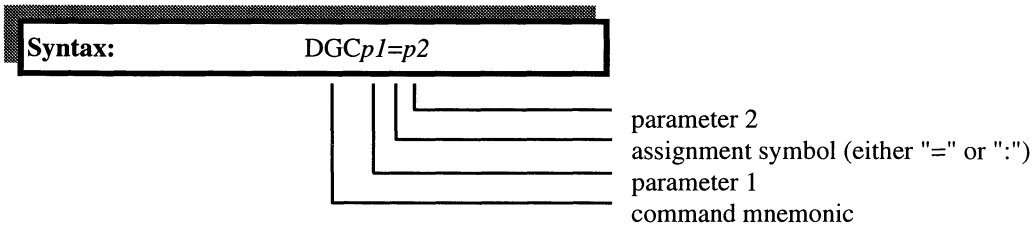
What the command does and how it is used

Any Restrictions on where the command can be used

Additional Remarks on the use of the command

A short program Example showing how the command is used

Any commands that are related to the command being described on the page



In addition, the controller address (0 through 9 or A through V) should always come before the command (this is automatically done in CCS), and a carriage return is used to end the command.

Command Summary

System	*mnemonic*	*description*
	FMW	downloads firmware
	RVM	retrieves user memory
	SVM	saves user memory
	CLM	clears all user memory and resets parameters to factory defaults
	SF	secures user memory space
	ATE	enables auto retrieving of user memory
	FP	assigns floating point variable allocation
	BD	assigns baud rate of serial port
	HSE	enables XON, XOFF handshake protocol on serial port
	CIE	enables computer interface format on serial port
	DSE	enables display format on serial port
	KY	puts character into key buffer
	EKB	empties key buffer
	PW	prompts for password
	PWC	prompts for password change
	RSF	resets faults
	STF	sets fault
	SRTC	sets real time clock

Axis	*mnemonic*	*description*
	PFE	enables position feedback
	PWE	enables position register wrap
	FO	loads forward software overtravel limit
	RO	loads reverse software overtravel limit
	MD	assigns direction of motor for forward moves
	AUR	loads axis unit ratio
	APL	loads axis position length
	FB	loads following error bound
	PB	loads in-position band
	FR	loads axis feedback resolution
	AR	loads amplitude of resolver excitation
	PR	loads motor poles to resolver poles ratio
	CO	loads commutation angle offset
	CA	loads commutation angle advance
	CC	loads continuous current in percent of maximum
	PC	loads peak current in percent of maximum
	SC	loads power save current percentage
	TC	loads torque limit current in percent of maximum
	TLE	enables torque limit
	POE	enables power output stage
	KP	loads proportional control gain
	KI	loads integral control gain
	KD	loads derivative control gain
	KA	loads acceleration feedforward
	FT	loads filter time constant
	CT	loads correction time constant
	BL	loads backlash constant
	DB	loads deadband constant

	PF	assigns position feedback input type
	MS	automatically sets up motor constants
	KS	automatically sets up control constants
	PFD	loads position feedback denominator
	PFN	loads position feedback numerator
	XFE	enables auxiliary position feedback
	XI	assigns auxiliary input type
	XPL	loads auxiliary position length
	XUR	loads auxiliary unit ratio

Program	*mnemonic*	*description*
	DE	defines program
	DEL	deletes current statement in program/motion block editor
	L	makes last statement the current statement in program/motion block editor
	X	steps through program/motion block
	LB	makes statement at label the current statement in the program editor
	!	exits program/motion block editor
	FLT	enters program/motion block editor and makes statement which faulted system current statement
	/	comment
	ED	ends program and exits program editor
	GT	unconditionally "gotos" label
	GS	unconditionally "gosubs" label
	RET	returns from "gosub"
	POP	pops "gosub" address from top of "gosub" stack
	RST	resets "gosub" stack to empty
	EX	executes program
	KL	kills program
	KLA	kills all programs
	SUP	suspends motion
	RSM	resumes motion
	LK	locks interpreter to program
	ULK	unlocks interpreter from program
	EXSV	executes command stored in string variable
	MB	defines motion block
	MED	ends motion block and exits motion block editor
	MEX	executes motion block
	MRP	repeats motion from beginning of motion block
	STM	loads start time of timer and starts timer
	WT	waits for expression to be true
	WT..ON..GT	waits for expression to be true or on condition becoming true "gotos" label
	IF..GT	conditionally "gotos" label
	IF..GS	conditionally "gosubs" label
	STB...GT	sets boolean variable and if variable wasn't set "gotos" label

Register	*mnemonic*	*description*
	AM	loads absolute move position
	OM	loads offset move position
	IM	loads incremental move position
	MP	assigns motion parameter
	JK	loads jerk percentage

	AC	loads acceleration & deceleration
	DC	loads deceleration
	VL	loads velocity
	ACP	loads acceleration & deceleration percentage
	DCP	loads deceleration percentage
	MVT	loads move time
	SAP	sets axis position register
	SOP	sets offset position register
	IOP	increments offset position register
	PHB	loads phase error bound
	PHG	loads phase gain
	PHL	loads phase length
	PHO	loads phase offset
	PHT	loads phase lockout time
	PHZ	loads phase zero
	RD	loads ratio denominator
	RN	loads ratio numerator
	SPH	sets phase position to zero
	SXP	sets auxiliary position register

Motion	*mnemonic*	*description*
	HT	halts all motion
	ST	stops all motion
	RAN	runs to absolute move position
	RON	runs to offset move position
	RIN	runs to incremental move position
	RFN	runs forward
	RRN	runs reverse
	HFH	homes forward to home input
	HRH	homes reverse to home input
	HFO	homes forward to overtravel input
	HRO	homes reverse to overtravel input
	HFM	homes forward to marker
	HRM	homes reverse to marker
	SI	applies step input
	PHE	enables phase locked loop
	RIE	enables ratio input

In/Out	*mnemonic*	*description*
	RSD	resets digital output
	RSDA	resets all digital outputs
	STD	sets digital output
	STDA	sets all digital outputs
	TGD	toggles digital output
	DO	sets and resets digital outputs according to number
	DFT	loads digital input filter time
	DIF	assigns filter to digital input
	OFE	enables fault on output fault
	DOE	enables digital outputs
	AOE	enables analog output
	AO	loads analog output with number/assigns signal to analog output
	SPA*n*B	loads set point A beginning

	SPB*n*B	loads set point B beginning
	SPA*n*E	loads set point A ending
	SPB*n*E	loads set point B ending
	SPA*n*F	forgets set point A
	SPB*n*F	forgets set point B
	PRT	prints string to display with carriage return and line feed
	PNT	prints string to display without carriage return and line feed
	ASC	sends ASCII code to display
	GET	gets one character from key buffer
	IN	inputs variable value from key buffer
	OUT	outputs variable value to display
	FW	assigns field width of output variable on display
	DP	assigns number of places after decimal point on display
	ADB	loads analog input deadband
	AOF	loads analog input offset

Variable	*mnemonic*	*description*
	RSB	resets boolean variable
	STB	sets boolean variable
	BV	loads boolean variable
	IV	loads integer variable
	IVP	loads integer variable pointed to by another integer variable
	DV	loads double integer variable with double expression
	FV	loads floating point variable
	FVP	loads floating point variable pointed to by integer variable
	SV	loads string variable

Diagnostic	*mnemonic*	*description*
	REV	reports software revision
	RMR	reports memory remaining
	RBV	reports boolean variable value
	RIV	reports integer variable value
	RFV	reports floating point variable value
	RSV	reports string variable value
	RDO	reports digital output
	RDI	reports digital input
	REG	reports positive edge sensitive digital input
	RAO	reports analog output
	RTC	reports real time clock value
	RTM	reports timer value
	RPS	reports program status
	RSRS	reports system status register
	RSRA	reports axis status register
	RIO	reports I/O register
	RFI	reports fault input register
	RFC	reports fault code
	RCP	reports command position
	RAP	reports axis position
	RAT	reports axis position capture
	RAV	reports axis velocity
	RUC	reports control output
	RFE	reports following error

RRR	reports resolver reading
DGE	enables diagnostics
DGL	prints diagnostic line of items
DGP	prints diagnostic message to terminal
DGS	sets program to single step mode
DGC	assigns diagnostic condition for printing
DGI	assigns diagnostic item to be printed
RAI	reports analog input
RPE	reports axis phase error
RPM	reports axis phase multiplier
RPP	reports axis phase position
RXP	reports auxiliary position
RXT	reports auxiliary position capture
RXV	reports auxiliary velocity

! EXITS PROGRAM/MOTION BLOCK EDITOR

Type: program

Syntax: !

Usage: This command exits the program or motion block editor.

Restrictions: not allowed out of programs or motion blocks

Example:

```
DE1                 (define program 1)
*  SAP:0
X                   (step through program)
*  AC=10
!                   (exit program editor)
*
```

See Also: DE
ED
MED

/ COMMENT

Type: program

Syntax: */ p1*

Parameters:

	p1
type	string
minimum	0 characters
maximum	127 characters

Usage: This command is used to add textual comments to a program or motion block.

Restrictions: not allowed out of programs or motion blocks

Remarks: Comments are stored as part of a program or motion block, but are ignored when the program or motion block is executing.

Example:

```
DE1                          (define program number 1)
/Set position register to 0
SAP:0
```

AC — LOADS ACCELERATION & DECELERATION

Type: register

Syntax: AC=*p1*

Parameters:

	p1
type	vexpr
units	units/sec^2
default	100 pulses/sec^2
minimum	100 pulses/sec^2
maximum	1,000,000,000 pulses/sec^2

Usage: This command loads both an acceleration and deceleration rate. The deceleration rate can be defined separately using the command DC. In cases where the acceleration rate differs from the deceleration rate, you must load AC first and DC second. AC is used when the motion parameter, MP, is assigned to velocity.

Remarks: The numerical values for the default, minimum and maximum of the parameter *p1* shown above are assuming that the axis unit ratio, AUR, is set at its default value of 1. If the axis unit ratio is set to a value other than 1, the default, maximum and minimum values will change appropriately (see AUR).

Example:

SAP:0	(set axis position register)
VL=10	(load velocity)
AC=40	(load acceleration)
AM=12	(load absolute move position)
RAN	(run to absolute move position)

What will happen:

By setting the axis position register, velocity, acceleration, absolute move position and issuing the RAN command, the axis will move 12 units in the forward direction. It will accelerate at 40 units/sec^2 to a velocity of 10 units/sec, and then decelerate at 40 units/sec^2 to zero velocity.

See Also: DC, ACP, MP, AUR

ACP — LOADS ACCELERATION & DECELERATION PERCENTAGE

Type: register

Syntax: ACP=*p1*

Parameters:

	p1
type	vexpr
units	%
default	50
minimum	1
maximum	99

Usage: This command loads both an acceleration and deceleration percentage. The deceleration percentage can be defined separately by using the command DCP. In cases where the acceleration percentage differs from the deceleration percentage, you must load ACP first and DCP second. The acceleration percentage is defined as the percentage of move time that the axis will accelerate. The deceleration percentage is similarly defined (see DCP). ACP is used when the motion parameter, MP, is assigned to time.

Remarks: 1. If ACP is set to a value greater than 50, then DCP is automatically set to the value of ACP taken away from 100. For example, if ACP is set to 70, DCP will be set to 30. Otherwise, DCP=ACP. 2. If ACP and DCP are assigned separately, their values cannot be set so that ACP+DCP>100.

Example:

IM=5	(load incremental move position)
MP=T	(assign motion parameter to time)
MVT=10	(load move time)
ACP=40	(load acceleration percentage)
RIN	(run to incremental move position)

What will happen:

By setting the incremental move position, move time, acceleration percentage and issuing the RIN command, the axis will move 5 units in the forward direction in 10 seconds. It will accelerate 40% of the move time (i.e., 4 seconds), then stay at a constant speed for 20% of move time (i.e., 2 seconds), then decelerate for the last 40% of move time (i.e., 4 seconds).

See Also: DCP
AC
MP
MVT

ADB — LOADS ANALOG INPUT DEADBAND

Type: input/output

Syntax: ADB=*p1*

Parameters:

p1	
type	vexpr
units	volts
default	0
minimum	0
maximum	10.000

Usage: This commnand loads the analog input deadband. The analog input deadband defines a range over which the analog input remains constant at 0 volts. When the analog input is less than or equal to ADB, the analog input is set to 0.

Restrictions: not allowed in motion blocks; extended command set only

See Also: RAI

AM — LOADS ABSOLUTE MOVE POSITION

Type: register

Syntax: AM=*p1*

Parameters:

	p1
type	vexpr
units	units
default	0 pulses
minimum	-2,000,000,000 pulses
maximum	2,000,000,000 pulses

Usage: This command loads the absolute move position register.

Remarks: The numerical values for the default, minimum and maximum of the parameter *p1* shown above are assuming that the axis unit ratio, AUR, is set at its default value of 1. If the axis unit ratio is set to a value other than 1, the default, maximum and minimum values will change appropriately (see AUR).

Example:

SAP:0	(set axis position register)
VL=10	(load velocity)
AC=40	(load acceleration rate)
AM=8	(load absolute move position)
RAN	(run to absolute move position)

What will happen:

By setting the axis position register, velocity, acceleration, absolute move position and issuing the RAN command, the axis will move 8 units in the forward direction.

See Also: RAN
IM
OM
AUR

AO — LOADS ANALOG OUTPUT WITH NUMBER/ ASSIGNS SIGNAL TO ANALOG OUTPUT

Type: input/output

Syntax: AO=*p1*

Parameters:

p1

type	vexpr **or** signal type
units	volts
default	V
minimum	-10.000
maximum	10.000

Usage: This command is used to either:

1. load the analog output with a certain number, or
2. assign a signal to the analog output, where
   V=velocity of axis (10 volts = 20 Krpm),
   C=control output (10 volts = maximum peak rating of drive),
   F=following error (10 volts = 2048 pulses of following error).

Remarks: The analog output must be enabled to output the value assigned. If the analog output is disabled, the output will be zero.

Example:

AO=V	(assign axis velocity to analog output)
AO=2	(load number into analog output)

See Also: AOE
RAO

AOE — ENABLES ANALOG OUTPUT

Type: input/output

Syntax: AOE=*p1*

Parameters:

p1	
type	boolean **or** bexpr
default	0
allowed values	0, 1

Usage: This command is used to enable the analog output. When AOE is set to 1, the output is enabled, and when AOE is set to 0, the output is disabled.

Restrictions: not allowed in motion blocks

Remarks: The analog output is disabled upon power up.

See Also: AO

AOF — LOADS ANALOG INPUT OFFSET

Type: input/output

Syntax: AOF=*p1*

Parameters:

	p1
type	vexpr
units	volts
default	0
minimum	-10.000
maximum	10.000

Usage: This command loads the analog input offset. This is used to add a voltage offset to the analog input.

Restrictions: not allowed in motion blocks; extended command set only

See Also: RAI

APL — LOADS AXIS POSITION LENGTH

Type: axis

Syntax: APL=*p1*

Parameters:

p1	
type	integer
units	units
default	2,000,000,000 pulses
minimum	500 pulses
maximum	2,000,000,000 pulses

Usage: This command loads the axis position length. This is actually half the axis position register length. The axis position register will count from -APL units to APL-1 units if position register wrap, PWE, is enabled. APL has no effect on the axis position register if PWE is disabled.

Restrictions: not allowed in programs or motion blocks

Remarks: The numerical values for the default, minimum and maximum of the parameter *p1* shown above are assuming that the axis axis unit ratio, AUR, is set at its default value of 1. If the axis axis unit ratio is set to a value other than 1, the default, maximum and minimum values will change appropriately (see AUR).

See Also: PWE
AUR

AR — LOADS AMPLITUDE OF RESOLVER EXCITATION

Type: axis

Syntax: AR=*p1*

Parameters:

p1	
type	vexpr
default	1
minimum	1
maximum	4

Usage: This command loads the amplitude of the signal needed for resolver excitation. The value of AR is determined by the transformation ratio of the resolver. The table below lists the possibilities:

p1	transformation ratio
1	0.50
2	1.00
3	reserved
4	reserved

Restrictions: resolver feedback servo only

ASC — SENDS ASCII CODE TO DISPLAY

Type: input/output

Syntax: ASC*p1*

Parameters:

p1	
type	integer **or** integer variable **or** pointed integer variable
minimum	0
maximum	255

Usage: This command sends one character represented by one of the ASCII codes 0 through 255, to a display attached to the serial port.

Remarks: If display format is disabled, the character will be sent to the terminal.

Examples:

ASC65	(send "A" to display)
IV1=13	(load integer variable)
ASCIV1	(send carriage return to display)
IV1=55	(load integer variable with pointer)
IV55=10	(load integer variable)
ASCIVP1	(send line feed to display)

See Also: DSE
OUT
PRT
PNT

ATE — ENABLES AUTO RETRIEVING OF USER MEMORY

Type: system

Syntax: ATE=*p1*

Parameters:

p1	
type	boolean
default	0
allowed values	0, 1

Usage: This command is used to enable auto retrieving of user memory from non-volatile memory on power up. If ATE is set to 1, auto retrieve is enabled, and if set to 0, auto retrieve is disabled.

Restrictions: not allowed in programs or motion blocks

Remarks: The value of ATE is checked in non-volatile memory upon power up. Therefore, once the value of ATE has been changed, the SVM command must be executed so that the value of ATE can be stored in non-volatile memory.

See Also: SVM

AUR — LOADS AXIS UNIT RATIO

Type: axis

Syntax: AUR=*p1*

Parameters:

	p1
type	integer
units	pulses/unit
default	1
minimum	1
maximum	1,000,000

Usage: This command loads the axis unit ratio.

Restrictions: not allowed in programs or motion blocks

Remarks: The numerical values for the default, minimum and maximum of all parameters whose units are pulses are assuming that the axis unit ratio, AUR, is set at its default value of 1. If the axis unit ratio is set to a value other than 1, the maximum and minimum values will be divided by the axis unit ratio. For instance, if the maximum value of a parameter is 2,000,000,000 pulses and the axis unit ratio is set to 4096, the new maximum of that parameter will be (2,000,000,000 pulses)/(4096 pulses/unit) = 488,281 units.

BD — ASSIGNS BAUD RATE OF SERIAL PORT

Type: system

Syntax: BD=*p1*

Parameters:

p1	
type	integer
units	baud (bits/sec)
default	set by DIP switch
allowed values	1200, 9600, 19200, 38400

Usage: This command assigns the baud rate of the serial port to the controller.

Restrictions: not allowed in motion blocks

Remarks: The baud rate is set by the DIP switch on the controller upon power up.

BL — LOADS BACKLASH CONSTANT

Type: axis

Syntax: BL=*p1*

Parameters:

	p1
type	vexpr
units	pulses
default	0
minimum	0
maximum	16,000

Usage: This command loads the backlash constant, which is used to compensate for mechanical backlash when using encoder position feedback. Encoder position feedback is enabled by setting PFE equal to 1 for steppers, and by setting XFE equal to 1 for servos.

Restrictions: stepper or dual loop feedback servo only

See Also: CT
DB
PFE
XFE

BV — LOADS BOOLEAN VARIABLE

Type: variable

Syntax: BV*p1=p2*

Parameters:

	p1	*p2*
type	integer	bexpr
minimum	1	
maximum	256	
allowed values		0, 1

Usage: This command loads boolean variable *p1* with the result of the evaluated boolean expression *p2*. If you simply wish to set a boolean variable to 0 or 1, you must use the RSB (reset boolean variable) and/or STB (set boolean variable) commands.

Examples:

BV1=IV>0	(BV1 set if IV1 is greater than 0)
BV2=IV1<>5	(BV2 set if IV1 is not equal to 5)
BV3=BV1 AND BV2	(BV3 set if both BV1 and BV2 are set)

See Also: RSB
STB
RBV

CA — LOADS COMMUTATION ANGLE ADVANCE

Type: axis

Syntax: CA=*p1*

Parameters:

p1	
type	vexpr
units	degrees per 75,000 pulses/sec
default	0
minimum	-90.0
maximum	90.0

Usage: This command loads the commutation angle advance for the motor.

Restrictions: resolver feedback servo only

Remarks: The commutation angle advance compensates for the lag in the commutation angle at high speed introduced by the inductance of the motor.

See Also: CO

CC — LOADS CONTINUOUS CURRENT IN PERCENT OF MAXIMUM

Type: axis

Syntax: CC=*p1*

Parameters:

	p1
type	vexpr
units	%
default	100 (servo) **or** 60 (stepper)
minimum	1
maximum	100

Usage: This command loads the continuous current available for the motor in percent of the continuous current rating of the drive.

Remarks: Listed below are the values for CC that you should use depending on the servo motor and IMC unit you are using:

	IMC unit			
Motor	3 Amps	6 Amps	12 Amps	24 Amps
3S22-G	46	23	11	5
3S32-G	96	48	24	12
3S33-G	100	53	26	13
3S33-H	100	100	53	26
3S34-G	100	50	25	12
3S35-G	96	48	24	12
3S43-G	96	48	24	12
3S43-H	100	93	46	23
3S45-G	100	91	45	22
3S45-H	100	100	91	45
3S46-G	100	91	45	22
3S46-H	100	100	91	45
3S63-G	100	100	91	45
3S65-G	100	100	89	44
3S67-G	100	100	94	47
3S88-G	100	100	100	100
3S8A-G	100	100	100	100

See Also: PC
SC
TC

CIE — ENABLES COMPUTER INTERFACE FORMAT ON SERIAL PORT

Type: system

Syntax: CIE=*p1*

Parameters:

p1	
type	boolean **or** bexpr
default	0
allowed values	0, 1

Usage: This command is used to change the serial port format to computer interface format. If CIE is set to 1, computer interface format is enabled, and if set to 0, computer interface format is disabled.

Restrictions: not allowed in motion blocks

CLM — CLEARS ALL USER MEMORY AND RESETS PARAMETERS TO FACTORY DEFAULTS

Type: system

Syntax: CLM

Usage: This command removes all programs and motion blocks and resets all parameters to default values.

Restrictions: not allowed in programs or motion blocks

Remarks:

1. This command is irreversible, that is, you cannot retrieve any programs, motion blocks or parameters that you have previously set.
2. This command will execute only when the controller is faulted and no programs or motion blocks are executing.

CO — LOADS COMMUTATION ANGLE OFFSET

Type: axis

Syntax: CO=*p1*

Parameters:

	p1
type	vexpr
units	degrees
default	-90.0
minimum	-180.0
maximum	180.0

Usage: This command loads the commutation angle offset of the motor.

Restrictions: resolver feedback servo only

Remarks: The commutation angle offset of the motor is set by the manufacturer. This value along with the value of PR can be set automatically by the MS command.

See Also: CA

CT — LOADS CORRECTION TIME CONSTANT

Type: axis

Syntax: CT=*p1*

Parameters:

p1	
type	vexpr
units	seconds
default	.01
minimum	.001
maximum	4.000

Usage: This command loads the correction time constant, which is the time that the system waits between position corrections when using encoder position feedback. Encoder position feedback is enabled by setting PFE equal to 1 for steppers, and by setting XFE equal to 1 for servos.

Restrictions: stepper or dual loop feedback servo only

See Also:
BL
DB
PFE
XFE

DB — LOADS DEADBAND CONSTANT

Type: axis

Syntax: DB=*p1*

Parameters:

p1	
type	vexpr
units	units
default	0 pulses (servo) **or** 10 pulses (stepper)
minimum	0 pulses
maximum	16,000 pulses

Usage: This command loads the deadband constant. This is the amount of position error allowed before the controller corrects the position error when using encoder position feedback. Encoder position feedback is enabled by setting PFE equal to 1 for steppers, and by setting XFE equal to 1 for servos.

Restrictions: stepper or dual loop feedback servo only

Remarks: The numerical values for the default, minimum and maximum of this register are assuming that the axis unit ratio, AUR, is set at its default value of 1. If the axis unit ratio is set to a value other than 1, the default, maximum and minimum values will change appropriately (see AUR).

See Also:
AUR
BL
CT
PFE
XFE

DC — LOADS DECELERATION

Type: register

Syntax: DC=*p1*

Parameters:

p1	
type	vexpr
units	units/sec^2
default	100 pulses/sec^2
minimum	100 pulses/sec^2
maximum	1,000,000,000 pulses/sec^2

Usage: This command loads the deceleration rate. In cases where the acceleration rate differs from the deceleration rate, you must load AC first and DC second. DC is used when the motion parameter, MP, is assigned to velocity.

Remarks: The numerical values for the default, minimum and maximum of the parameter *p1* shown above are assuming that the axis unit ratio, AUR, is set at its default value of 1. If the axis unit ratio is set to a value other than 1, the default, maximum and minimum values will change appropriately (see AUR).

Example:

SAP:0	(set axis position register)
VL=10	(load velocity)
AC=40	(load acceleration)
DC=10	(load deceleration)
AM=12	(load absolute move position)
RAN	(run to absolute move position)

What will happen:

By setting the axis position register, velocity, acceleration, deceleration, absolute move position and issuing the RAN command, the axis will move 12 units in the forward direction. It will accelerate at 40 units/sec^2 to a velocity of 10 units/sec, and then decelerate at 10 units/sec^2 to zero velocity.

See Also: AC, DCP, MP, AUR

DCP — LOADS DECELERATION PERCENTAGE

Type: register

Syntax: DCP=*p1*

Parameters:

p1	
type	vexpr
units	%
default	50
minimum	1
maximum	99

Usage: This command loads the deceleration percentage. In cases where the acceleration percentage differs from the deceleration percentage, you must load ACP first and DCP second. The deceleration percentage is defined as the percentage of move time that the axis will decelerate. DCP is used when the motion parameter, MP, is assigned to time.

Remarks: If ACP and DCP are assigned separately, their values cannot be set so that ACP+DCP>100.

Example:

IM=5	(load incremental move position)
MP=T	(assign motion parameter to time)
MVT=10	(load move time)
ACP=25	(load acceleration percentage)
DCP=40	(load deceleration percentage)
RIN	(run to incremental move position)

What will happen:

By setting the incremental move position, move time, acceleration percentage, deceleration percentage and issuing the RIN command, the axis will move 5 units in the forward direction in 10 seconds. It will accelerate 25% of the move time (i.e., 2.5 seconds), then stay at a constant speed for 35% of move time (i.e., 3.5 seconds), then decelerate for the last 40% of move time (i.e., 4 seconds).

See Also: ACP
DC
MP
MVT

DE — DEFINES PROGRAM

Type: program

Syntax: DE*p1*

Parameters:

p1	
type	integer
minimum	1
maximum	4

Usage: This command enters the program line editor at the first statement of program *p1*. It is used to either view programs or to start editing them.

Restrictions: not allowed in programs or motion blocks

Remarks: This command will execute only when the axis is stopped and no programs or motion blocks are executing.

Example:

```
DE1          (define program 1)
SAP:0        (set axis position register)
VL=10        (load velocity)
AC=40        (load acceleration rate)
AM=12        (load absolute move position)
RAN          (run to absolute move position)
ED           (end program 1 and exit program editor)
```

See Also: MB, ED, X, !, DEL, L, LB, FLT

DEL — DELETES CURRENT STATEMENT IN PROGRAM/ MOTION BLOCK EDITOR

Type: program

Syntax: DEL

Usage: This command deletes the current statement in the program or motion block editor and makes the next statement the current statement. It is used to edit programs or motion blocks.

Restrictions: not allowed out of programs or motion blocks

Example:

Change "AC=40" in program 1 (DE command example) to "AC=10":

```
DE1          (define program 1)
*  SAP:0
X            (step through program)
*  VL=10
X            (step through program)
*  AC=10
DEL          (delete current statement)
*  AM=12
AC=10        (load acceleration)
*  AM=12
!            (exit program editor)
```

See Also: DE
L
LB
X

DFT — LOADS DIGITAL INPUT FILTER TIME

Type: input/output

Syntax: DFT=*p1*

Parameters:

	p1
type	vexpr
units	seconds
default	.001
minimum	.001
maximum	4.000

Usage: This command loads the digital input filter time, which represents the minimum duration of a pulse that the filter will pass.

Restrictions: not allowed in motion blocks

Remarks: The primary use for this command is to debounce a contact connected to a digital input. Generally, contact bounce lasts for less than 30 milliseconds. So, setting DFT=.03 should debounce the contact.

Example:

DFT=.03	(sets digital input filter time to 30 ms)

See Also: DIF

DGC — ASSIGNS DIAGNOSTIC CONDITION FOR PRINTING

Type: diagnostic

Syntax: DGC*p1=p2*

Parameters:

	p1	*p2*
type	integer	bexpr
default		NULL
minimum	1	
maximum	4	
allowed values		any boolean expression, NULL

Usage: This command assigns diagnostic condition *p1*. When one of the user defined diagnostic conditions is satisfied, and if diagnostics are enabled, a diagnostic line of items is sent to the terminal (see DGL, DGI).

Restrictions: not allowed in programs or motion blocks

Remarks: Upon clearing the memory with the CLM command, all diagnostic conditions and items are set to the value "NULL", which means that there are no diagnostic conditions/items assigned. If you wish to eliminate the assignment of diagnostic condition *p1*, use the DGC command and set parameter *p2* to "NULL". For example, DGC1=NULL will eliminate the assignment of diagnostic condition 1.

Example:

```
STM2=0.5          (load start time of timer 2, start timer 2)
DGC1=TM2 AND PX1  (assign diagnostic condition 1)
```

What will happen:

By loading timer 2 and assigning diagnostic condition 1, when diagnostics are enabled, a diagnostic line of items will be sent to the terminal every 0.5 seconds while program 1 is executing. Each diagnostic line will begin with the diagnostic condition satisfied, which in this case would be "TM2 AND PX1", and then be followed by a colon and the diagnostic items loaded.

See Also: DGE, DGI, DGL, DGP

DGE — ENABLES DIAGNOSTICS

Type: diagnostic

Syntax: DGE=*p1*

Parameters:

p1	
type	boolean
default	0
allowed values	0, 1

Usage: This command is used to enable diagnostics for the controller. When DGE is set to 1, diagnostics are enabled, and when set to 0, diagnostics are disabled.

Restrictions: not allowed in programs or motion blocks

Remarks: Diagnostics are disabled upon power up.

See Also:
DGC
DGI
DGL
DGP
DGS

DGI — ASSIGNS DIAGNOSTIC ITEM TO BE PRINTED

Type: diagnostic

Syntax: DGI*p1*=*p2*

Parameters:

	p1	*p2*
type	integer	non-boolean operand **or** non-boolean parameter
default		NULL
minimum	1	
maximum	8	
allowed values		BV*n*, IV*n*, IVP*n*, FV*n*, FVP*n*, SV*n*, SVIV*n*, SVIVP*n*, DO, DI, EG, AO, RTC, TM*n*, PS*n*, SRS, SRA, IO, FI, FC, CP, AP, AT, AV, UC, FE, RR, FP, BD, AUR, APL, FO, RO, FB, PB, FR, AR, PR, CO, CA, CC, PC, SC, TC, KP, KI, KD, KA, FT, CT, BL, DB, STM*n*, AM, OM, IM, JK, AC, DC, VL, ACP, DCP, MVT, DFT, DIF, SPA*n*B, SPA*n*E, SPB*n*B, SPB*n*E, FW, DP, DGS, NULL

Usage: This command assigns a diagnostic item to be printed. It defines which diagnostic items will be printed out whenever a DGL is executed or one of the user defined diagnostic conditions is met.

Restrictions: not allowed in programs or motion blocks

Remarks: Upon clearing the memory with the CLM command, all diagnostic conditions and items are set to the value "NULL", which means that there are no diagnostic conditions/items assigned. If you wish to eliminate the assignment of diagnostic item *p1*, use the DGI command and set parameter *p2* to "NULL". For example, DGI1=NULL will eliminate the assignment of diagnostic item 1.

Example:

DGI1=AP	(assign diagnostic item 1)
DGI2=AV	(assign diagnostic item 2)
DGI3=FE	(assign diagnostic item 3)
DGI4=RR	(assign diagnostic item 4)

What will happen:

By assigning these diagnostic items, and if diagnostics are enabled, the diagnostic items will be sent to the terminal when the DGL command is executed.

See Also: DGE
DGC
DGL
DGP

DGL — PRINTS DIAGNOSTIC LINE OF ITEMS

Type: diagnostic

Syntax: DGL

Usage: This command prints a diagnostic line of items to the terminal that have been assigned with the DGI command. This only works while diagnostics are enabled.

Remarks: Since this command is ignored when diagnostics are not enabled, the command can be left in programs even when not using diagnostics.

Example:

```
DGI1=AP          (assign diagnostic item 1)
DGI2=AV          (assign diagnostic item 2)
DGI3=FE          (assign diagnostic item 3)
DGI4=RR          (assign diagnostic item 4)
DGL              (prints diagnostic line of items)
*DGL: AP=0, AV=0
DGL: FE=0, RR=3061
```

See Also: DGE
DGC
DGI
DGP

DGP — PRINTS DIAGNOSTIC MESSAGE TO TERMINAL

Type: diagnostic

Syntax: DGP"*p1*"

Parameters:

p1	
type	string
minimum	0 characters
maximum	127 characters

Usage: This command prints the diagnostic message *p1* to the terminal. It works only when diagnostics are enabled.

Remarks: Since this command is ignored when diagnostics are not enabled, the command can be left in programs even when not using diagnostics.

Example:

```
DGE=1                    (enable diagnostics)
DGP"Diagnostics
      enabled"           (send to diagnostic message to serial port)
*Diagnostics enabled
```

See Also: DGE
DGC
DGI
DGL

DGS — SETS PROGRAM TO SINGLE STEP MODE

Type: diagnostic

Syntax: DGS=*p1*

Parameters:

p1	
type	integer
default	0
minimum	0
maximum	4

Usage: This command sets program *p1* to single step mode. If DGS is set to 0, single step mode is disabled. Single step mode can only occur when diagnostics are enabled.

Restrictions: not allowed in motion blocks

Remarks: To execute a program while in single step mode, use the X command to step through the program, i.e. execute the program one statement at a time. As each line of the program is executed, it is sent to the terminal.

Example:

```
DGE=1          (enable diagnostics)
DGS=3          (set program 3 to single step mode)
EX3            (execute program)
* SAP:0
X              (step through program)
* VL=5
X              (step through program)
* AC=1
X
* IM=40
X
* RIN
X
* ED
X
*
```

What will happen:

By enabling diagnostics, setting program 3 to single step mode and executing program 3, only the first line of the program will be executed. Then, by using the X command, the program will execute the next line, send that line to the terminal, and so on until the end of the program is reached.

See Also: DGE

DIF — ASSIGNS FILTER TO DIGITAL INPUT

Type: input/output

Syntax: DIF=*p1*

Parameters:

p1	
type	12-bit binary number **or** vexpr
default	000000000000
minimum	000000000000
maximum	111111111111

Usage: This command tells the controller which digital inputs are to be filtered. The left-most bit of the 12-bit binary number represents input 12, and the right-most bit represents input 1. If an input's corresponding bit is set to 1, then the input is to be filtered, and if set to 0, it is not to be filtered.

Restrictions: not allowed in motion blocks

Remarks: To use vexpr as a parameter, use the decimal equivalent of the 12-bit binary number.

Examples:

DFT=000010010110	(assign filter to digital inputs 8, 5, 3 and 2)
IV1=150	(load integer variable 1)
DFT=IV1	(assign filter to digital inputs 8, 5, 3 and 2)

See Also: DFT

DO — SETS AND RESETS DIGITAL OUTPUTS ACCORDING TO NUMBER

Type: input/output

Syntax: DO:*p1*

Parameters:

p1	
type	vexpr
minimum	64
maximum	4,032

Usage: This command sets and resets digital outputs according to a number.

Remarks: To use vexpr as a parameter, use the decimal equivalent of the 12-bit binary number that represents which digital outputs are to be set/reset.

Example:

DO: 2368 (set digital outputs 12, 9 and 7 [NOTE: 2368_{10}=100101000000_2])

See Also:
DOE
RSD
RSDA
STD
STDA
RDO

DOE — ENABLES DIGITAL OUTPUTS

Type: input/output

Syntax: DOE=*p1*

Parameters:

	p1
type	boolean **or** bexpr
default	0
allowed values	0, 1

Usage: This command is used to enable the digital outputs. If DOE is set to 1, the digital outputs are enabled, and if it is set to 0, the digital outputs are disabled.

Restrictions: not allowed in motion blocks

Remarks: The digital outputs are disabled upon power up.

See Also:
DO
RSD
RSDA
STD
STDA

DP — ASSIGNS NUMBER OF PLACES AFTER DECIMAL POINT ON DISPLAY

Type: input/output

Syntax: DP=*p1*

Parameters:

p1	
type	vexpr
default	10
minimum	0
maximum	10

Usage: This command assigns the number of places after the decimal point of a floating point number on the display.

Remarks: This command only affects numbers that are sent to the display using the OUT command.

See Also: FW
OUT

DSE — ENABLES DISPLAY FORMAT ON SERIAL PORT

Type: system

Syntax: DSE=*p1*

Parameters:

p1	
type	boolean **or** bexpr
default	1
allowed values	0, 1

Usage: This command is used to enable the display format on the serial port. If DSE is set to 1, display format is enabled, and if set to 0, display format is disabled.

Restrictions: not allowed in motion blocks

Remarks: When display format is disabled, all textual output will be sent to the terminal. This means that the ASC, PRT, PNT, and OUT commands will send ASCII codes, strings and variable values to the terminal rather than the display.

See Also:
ASC
PRT
PNT
OUT

DV — LOADS DOUBLE INTEGER VARIABLE WITH DOUBLE EXPRESSION

Type: variable

Syntax: DV*p1*=*p2* DV*p1* *p3* DV*p1*

Parameters:

	p1	*p2*	*p3*	
type	integer	function	binary operator	double expression
minimum	1			-2^{63}
maximum	4095			$2^{63}-1$
allowed values	odd only	DABS, DSQR	D+, D-, D*, D/	64 bit signed number

Usage: This command loads double integer variable *p1* with a double expression.

Remarks:

1. Note that *p2* is optional, and *p3* with the last double integer variable is also optional.
2. For the square root operation, the operand can be at most a 62 bit positive number.
3. For the multiply operation, the right operand can be up to a 64 bit signed number, but the left operand can be at most a 32 bit signed number.
4. For the divide operation, the dividend can be up to a 64 bit signed number, but the divisor can be at most a 32 bit signed number.

Examples:

```
IV1=15000          (load integer variable 1 with 15,000)
IV3=325000         (load integer variable 3 with 325,000)
DV5=DV1 D+ DV3     (load double integer variable 5 with 340,000 [15,000 + 325,000])
DV7=DSQR DV1 D*
      DV5          (load double integer variable 7 with square root of 5,100,000,000 [340,000 *
                     15,000])
RIV7               (report integer variable 7 value)
*71414
```

See Also: IV

ED — ENDS PROGRAM AND EXITS PROGRAM EDITOR

Type: program

Syntax: ED

Usage: This command marks the end of a program and exits from the program editor.

Restrictions: allowed only in programs

Remarks: Be cautious when using this command, since it will delete all program statements that follow it. If you only wish to exit the program editor, use the ! command.

Example:

```
DE1     (define program 1)
SAP:0   (set axis position register)
VL=10   (load velocity)
AC=40   (load acceleration)
AM=12   (load absolute move position)
RAN     (run to absolute move position)
ED      (end program 1 and exit program editor)
```

See Also: !
MED

EKB EMPTIES KEY BUFFER

Type: system

Syntax: EKB

Usage: This command empties the key buffer.

Restrictions: not allowed in motion blocks

See Also: KY
IN

EX — EXECUTES PROGRAM

Type: program

Syntax: EX*p1*

Parameters:

p1	
type	integer **or** integer variable **or** pointed integer variable
minimum	1
maximum	4

Usage: This command executes program *p1*.

Restrictions: not allowed in motion blocks

Remarks: If program *p1* is already executing, then this command does nothing.

Example:

```
DE1       (define program 1)
SAP:0     (set axis position register)
VL=10     (load velocity)
AC=40     (load acceleration rate)
AM=12     (load absolute move position)
RAN       (run to absolute move position)
ED        (end program 1 and exit program editor)

EX1       (execute program 1)
```

What will happen:

After issuing the EX command, the axis will move 12 units in the forward direction.

See Also: MEX

EXSV — EXECUTE COMMAND STORED IN STRING VARIABLE

Type: program

Syntax: EXSV*p1*

Parameters:

p1	
type	integer **or** integer variable **or** pointed integer variable
minimum	1
maximum	16

Usage: This command executes the command stored in string variable *p1*.

Restrictions: not allowed in motion blocks

Remarks: Commands that are not allowed in programs cannot be executed using EXSV.

Example:

```
SV1="AM=10"     (load string variable 1)
EXSV1           (execute command stored in string variable 1)
```

What will happen:

By loading string variable 1 and executing the command stored in string variable 1, the absolute move position, AM, will be set to 10 units.

FB — LOADS FOLLOWING ERROR BOUND

Type: axis

Syntax: FB=*p1*

Parameters:

	p1
type	vexpr
units	units
default	400 pulses
minimum	0 pulses
maximum	16,000 pulses

Usage: This command loads the following error bound. This is a limit set on the following error, and if this limit is exceeded, the controller will fault.

Remarks: The numerical values for the default, minimum and maximum of the parameter *p1* shown above are assuming that the axis unit ratio, AUR, is set at its default value of 1. If the axis unit ratio is set to a value other than 1, the default, maximum and minimum values will change appropriately (see AUR).

See Also: AUR

FLT — ENTERS PROGRAM/MOTION BLOCK EDITOR AND MAKES STATEMENT WHICH FAULTED SYSTEM CURRENT STATEMENT

Type: program

Syntax: FLT

Usage: This command enters the program or motion block editor and makes the statement which faulted the system the current statement.

Restrictions: not allowed in programs or motion blocks

Remarks: This command will execute only when the axis is stopped and no programs or motion blocks are executing.

Example:

```
DE1          (define program 1)
SAP:0        (set axis position register)
STF          (set fault)
ED           (end program 1 and exit program editor)

EX1          (execute program 1)
FLT          (enter program editor and make statement which faulted system the current
              statement)
*STF
```

FMW — DOWNLOADS FIRMWARE

Type: system

Syntax: FMW

Usage: This command downloads the controller firmware and saves it in non-volatile memory.

Restrictions: not allowed in programs or motion blocks

Remarks: This command will execute only when the axis is stopped and no programs or motion blocks are executing.

FO — LOADS FORWARD SOFTWARE OVERTRAVEL LIMIT

Type: axis

Syntax: FO=*p1*

Parameters:

	p1
type	vexpr
units	units
default	2,000,000,000 pulses
minimum	-2,000,000,000 pulses
maximum	2,000,000,000 pulses

Usage: This command loads the forward software overtravel limit.

Remarks: The numerical values for the default, minimum and maximum of the parameter *p1* shown above are assuming that the axis unit ratio, AUR, is set at its default value of 1. If the axis unit ratio is set to a value other than 1, the default, maximum and minimum values will change appropriately (see AUR).

Example:

SAP:0	(set axis position register)
VL=10	(load velocity)
AC=40	(load acceleration)
AM=12	(load absolute move position)
FO=10	(load forward software overtravel limit)
RAN	(run to absolute move position)

What will happen:

By setting the axis position register, velocity, acceleration, absolute move position, forward software overtravel and issuing the RAN command, the axis will move 10 units in the forward direction and immediately halt all motion.

See Also: RO
AUR

FP — ASSIGNS FLOATING POINT VARIABLE ALLOCATION

Type: system

Syntax: FP=*p1*

Parameters:

p1	
type	integer
default	1024
minimum	0
maximum	2,048

Usage: This command assigns floating point variable allocation, i.e. allocates space in memory for *p1* floating point variables.

Restrictions: not allowed in programs or motion blocks

Remarks:

1. This command can be used only after the memory has been cleared using the CLM command. Floating point variable space cannot be reallocated.
2. In order to assign floating point variable allocation, this command has the effect of writing over part of the space normally allocated for integer variables. One floating point variable will take over the space that two integer variables were previously in. For example, if you set FP to 200, the integer variables will range from 1 to 4 ,096 - (2*200) = 3,796.

See Also: FV
FVP
IV

FR — LOADS AXIS FEEDBACK RESOLUTION

Type: axis

Syntax: FR=*p1*

Parameters:

	p1
type	vexpr
units	pulses/revolution
default	1,000 (encoder feedback servo) **or** 4,096 (resolver feedback servo)
minimum	500
maximum	1,000,000

Usage: This command loads the axis feedback resolution. This is defined as the number of feedback pulses per revolution of the axis.

Restrictions: servo only

FT — LOADS FILTER TIME CONSTANT

Type: axis

Syntax: FT=*p1*

Parameters:

p1	
type	vexpr
default	1
minimum	0
maximum	5

Usage: This command loads the filter time constant.

Restrictions: servo only

Remarks: The filter time constant is used to eliminate dither. Generally, the lower the bandwidth of a servo system, the higher the filter time constant should be. The equation for setting FT based on the torque to inertia ratio is:

$$FT = \left\lfloor \frac{120}{\sqrt{\frac{torque}{inertia}}} + 0.3 \right\rfloor$$

where the brackets mean to take the integer part of the number only. This value along with the values of all the other control constants can be set automatically by the KS command.

See Also: KS

FV — LOADS FLOATING POINT VARIABLE

Type: variable

Syntax: FV*p1*=*p2*

Parameters:

	p1	*p2*
type	integer	vexpr
minimum	1	1.5 x 10^{-39} (absolute value)
maximum	2048	1.7 x 10^{38} (absolute value)

Usage: This command loads floating point variable *p1* with result of the evaluated variable expression *p2*.

Remarks: The numerical value for the maximum of the parameter *p1* shown above is assuming that the floating point variable allocation, FP, is set at 2048. If the floating point variable allocation is set to a value other than 2048, the maximum value will change to the value loaded in FP.

Examples:

FV1=100	(load floating point variable 1 with 100.)
FV2=5.563	(load floating point variable 2 with 5.563)
FV3=SQR 2*FV2	(load floating point variable 3 square root of 11.126 [2*5.563])
FV4=AP/5	(load floating point variable 4 with axis position/5)

See Also: FVP
FP
RFV

FVP — LOADS FLOATING POINT VARIABLE POINTED TO BY INTEGER VARIABLE

Type: variable

Syntax: FVP*p1*=*p2*

Parameters:

	p1	*p2*
type	integer	vexpr
minimum	1	1.5×10^{-39} (absolute value)
maximum	4096	1.7×10^{38} (absolute value)

Usage: This command loads the floating point variable which is pointed to by integer variable *p1* with the result of the evaluated variable expression *p2*.

Remarks: The numerical value for the maximum of the parameter *p1* shown above is assuming that the floating point variable allocation, FP, is set at its default value of 0. If the floating point variable allocation is set to a value other than 0, the maximum value will change appropriately (see FP).

Example:

```
IV2=35          (load integer variable with pointer)
FVP2=10.5       (load floating point variable)
```

What will happen:

Floating point variable number 35, which is pointed to by integer variable 2, will be loaded with the value 10.5.

See Also: FV, FP, RFV

FW — ASSIGNS FIELD WIDTH OF OUTPUT VARIABLE ON DISPLAY

Type: input/output

Syntax: FW=*p1*

Parameters:

p1	
type:	vexpr
default	30
minimum	0
maximum	30

Usage: This command assigns the field width of a variable on the display.

Remarks:

1. If FW is set to 0, then numbers will be displayed without a field width. Instead, they will be displayed to maximum precision.
2. This command only affects numbers that are sent to the display using the OUT command.

See Also: DP
OUT

GET — GETS ONE CHARACTER FROM KEY BUFFER

Type: input/output

Syntax: GET *p1*

Parameters:

	p1
type	variable
allowed values	IV*n*, IVP*n*, FV*n*, FVP*n*, SV*n*, SVIV*n*, SVIVP*n*

Usage: This command gets one character from the key buffer and loads it into the variable *p1*.

Restrictions: allowed only in programs

Remarks: Using different variables will produce different results:

1. The ASCII code of the character put into the key buffer will be loaded into IV*n*, IVP*n*, FV*n*, or FVP*n*;
2. The character put into the key buffer will be loaded into SV*n*, SVIV*n* or SVIVP*n*.

Example:

```
DE1          (define program 1)
GET IV1      (get one character from the key buffer)
GET SV1      (get one character from the key buffer)
ED           (end program 1 and exit program editor)

EX1          (execute program 1)
KYE          (send one character to key buffer)
RIV1         (report integer variable value)
* 69
RSV1         (report string variable value)
* E
```

See Also: IN
OUT

GS — UNCONDITIONALLY "GOSUBS" LABEL

Type: program

Syntax: GS*p1*

Parameters:

	p1
type	integer **or** integer variable **or** pointed integer variable
minimum	1
maximum	999

Usage: This command causes the program to unconditionally go to the subroutine at label *p1*. The program will return when it encounters the RET command.

Restrictions: allowed only in programs

Remarks: There can be up to 32 nested gosub statements in a program.

Example:

```
     DE1                  (define program 1)
     SAP:0                (set axis position register)
     VL=1                 (load velocity)
     AC=10                (load acceleration)
     RFN                  (run forward)
     GS5                  (unconditionally gosub 5)
     IV1=6                (load integer variable)
     GSIV1                (unconditionally gosub 6)
     GT10                 (unconditionally goto 10)
5    PRT "Press any key to
          stop axis"      (print string to display with carriage return and line feed)
     GET IV2              (get one character from key buffer)
     ST                   (stop all motion)
     RET                  (return from gosub)
6    PNT "Axis position is "  (print string to display without carriage return and line feed)
     FV1=AP               (load floating point variable)
     OUT FV1              (output variable value to display)
     PRT " units."        (print string to display with carriage return and line feed)
     RET                  (return from gosub)
10   ED                   (end program 1 and exit program editor)
```

What will happen:

This program, once executed, runs the axis in the forward direction. It then goes to the subroutine at label 5, which waits for a character from the key buffer and returns upon receiving the character. Next, it goes to the subroutine at label 6, which prints the axis position on the display and returns. Lastly, it goes to the statement at label 10, which ends the program.

See Also: GT
RET
POP
RST

GT — UNCONDITIONALLY "GOTOS" LABEL

Type: program

Syntax: GT*p1*

Parameters: *p1*

type	integer **or** integer variable **or** pointed integer variable
minimum	1
maximum	999

Usage: This command causes the program to unconditionally go to the statement at label *p1*.

Restrictions: allowed only in programs

Example:

```
    DE1                     (define program 1)
    SAP:0                   (set axis position register)
    VL=1                    (load velocity)
    AC=10                   (load acceleration)
    RFN                     (run forward)
    GS5                     (unconditionally gosub 5)
    IV1=6                   (load integer variable)
    GSIV1                   (unconditionally gosub 6)
    GT10                    (unconditionally goto 10)
5   PRT "Press any key to
          stop axis"        (print string to display with carriage return and line feed)
    GET IV2                 (get one character from key buffer)
    ST                      (stop all motion)
    RET                     (return from gosub)
6   PNT "Axis position is " (print string to display without carriage return and line feed)
    FV1=AP                  (load floating point variable)
    OUT FV1                 (output variable value to display)
    PRT " units."           (print string to display with carriage return and line feed)
    RET                     (return from gosub)
10  ED                      (end program 1 and exit program editor)
```

What will happen:

This program, once executed, runs the axis in the forward direction. It then goes to the subroutine at label 5, which waits for a character from the key buffer and returns upon receiving the character. Next, it goes to the subroutine at label 6, which prints the axis position on the display and returns. Lastly, it goes to the statement at label 10, which ends the program.

See Also: GS

HFH — HOMES FORWARD TO HOME INPUT

Type: motion

Syntax: HFH

Usage: This command homes forward to the home input.

Restrictions: not allowed in motion blocks

Remarks: When this command is executed, the axis will run at a fixed velocity of 4,096 pulses/sec until the home input is encountered. The software overtravel limits, FO and RO, are ignored while homing the axis.

Example:

```
DE1     (define program 1)
HFH     (home forward to home input)
WT IP   (wait for axis to be in position)
SAP:0   (set axis position register)
ED      (end program 1 and exit program editor)
```

What will happen:

This program, once executed, will run the axis in the forward direction until the home input is encountered, wait for the axis to be in position, and then set the axis position register to 0.

See Also: HRH
HFM
HFO

HFM — HOMES FORWARD TO MARKER

Type: motion

Syntax: HFM

Usage: This command homes forward to the marker. The marker is defined as the zero position on the resolver of resolver feedback units, or the encoder channel index input on encoder feedback units.

Restrictions: not allowed in motion blocks

Remarks: When this command is executed, the axis will run at a fixed velocity of 4,096 pulses/sec until the marker is encountered. The software overtravel limits, FO and RO, are ignored while homing the axis.

Example:

```
DE1     (define program 1)
HFM     (home forward to marker)
WT IP   (wait for axis to be in position)
SAP:0   (set axis position register)
ED      (end program 1 and exit program editor)
```

What will happen:

This program, once executed, will run the axis in the forward direction until the marker is encountered, wait for the axis to be in position, and then set the axis position register to 0.

See Also: HRM
HFH
HFO

HFO — HOMES FORWARD TO OVERTRAVEL INPUT

Type: motion

Syntax: HFO

Usage: This command homes forward to the forward overtravel input.

Restrictions: not allowed in motion blocks

Remarks: When this command is executed, the axis will run at a fixed velocity of 4,096 pulses/sec until the forward overtravel input is encountered. The software overtravel limits, FO and RO, are ignored while homing the axis.

Example:

```
DE1       (define program 1)
HFO       (home forward to overtravel input)
WT IP     (wait for axis to be in position)
SAP:0     (set axis position register)
ED        (end program 1 and exit program editor)
```

What will happen:

This program, once executed, will run the axis in the forward direction until the forward overtravel input is encountered, wait for the axis to be in position, and then set the axis position register to 0.

See Also: HRO
HFH
HFM

HRH — HOMES REVERSE TO HOME INPUT

Type: motion

Syntax: HRH

Usage: This command homes reverse to the home input.

Restrictions: not allowed in motion blocks

Remarks: When this command is executed, the axis will run at a fixed velocity of 4,096 pulses/sec until the home input is encountered. The software overtravel limits, FO and RO, are ignored while homing the axis.

Example:

```
DE1     (define program 1)
HRH     (home reverse to home input)
WT IP   (wait for axis to be in position)
SAP:0   (set axis position register)
ED      (end program 1 and exit program editor)
```

What will happen:

This program, once executed, will run the axis in the reverse direction until the home input is encountered, wait for the axis to be in position, and then set the axis position register to 0.

See Also: HFH
HRM
HRO

HRM — HOMES REVERSE TO MARKER

Type: motion

Syntax: HRM

Usage: This command homes reverse to the marker. The marker is defined as the zero position on the resolver of resolver feedback units, or the encoder channel index input on encoder feedback units.

Restrictions: not allowed in motion blocks

Remarks: When this command is executed, the axis will run at a fixed velocity of 4,096 pulses/sec until the marker is encountered. The software overtravel limits, FO and RO, are ignored while homing the axis.

Example:

```
DE1       (define program 1)
HRM       (home reverse to marker)
WT IP     (wait for axis to be in position)
SAP:0     (set axis position register)
ED        (end program 1 and exit program editor)
```

What will happen:

This program, once executed, will run the axis in the reverse direction until the marker is encountered, wait for the axis to be in position, and then set the axis position register to 0.

See Also: HFM
HRH
HRO

HRO — HOMES REVERSE TO OVERTRAVEL INPUT

Type: motion

Syntax: HRO

Usage: This command homes reverse to the reverse overtravel input.

Restrictions: not allowed in motion blocks

Remarks: When this command is executed, the axis will run at a fixed velocity of 4,096 pulses/sec until the reverse overtravel input is encountered. The software overtravel limits, FO and RO, are ignored while homing the axis.

Example:

```
DE1       (define program 1)
HRO       (home reverse to overtravel input)
WT IP     (wait for axis to be in position)
SAP:0     (set axis position register)
ED        (end program 1 and exit program editor)
```

What will happen:

This program, once executed, will run the axis in the reverse direction until the reverse overtravel input is encountered, wait for the axis to be in position, and then set the axis position register to 0.

See Also: HFO
HRH
HRM

HSE — ENABLES XON, XOFF HANDSHAKE PROTOCOL ON SERIAL PORT

Type: system

Syntax: HSE=*p1*

Parameters:

p1	
type	boolean **or** boolean variable
default	0
allowed values	0, 1

Usage: This command is used to enable the XON, XOFF handshake protocol on the serial port. If HSE is set to 1, then handshake protocol is enabled, and if HSE is set to 0, then it is disabled.

Restrictions: not allowed in motion blocks

HT — HALTS ALL MOTION

Type: motion

Syntax: HT

Usage: This command immediately halts all axis motion.

Remarks: This command should only be used at low velocities or in extreme situations as the sudden stop may damage mechanical components in the system.

Example:

VL=10	(load velocity)
AC=10	(load acceleration)
RFN	(run forward)
HT	(halt all motion)

What will happen:

By setting the velocity and acceleration and issuing the RFN command, the axis will run in the forward direction. By issuing the HT command, the axis will immediately halt.

See Also: ST

IF...GS CONDITIONALLY "GOSUBS" LABEL

Type: program

Syntax: IF *p1* GS*p2*

Parameters:

	p1	*p2*
type	bexpr	integer **or** integer variable **or** pointed integer variable
minimum		1
maximum		999
allowed values	0, 1	

Usage: This command causes the program to conditionally go to the subroutine at label *p2* if *p1* is true (i.e., evaluates to 1). The program will return when it encounters the RET command.

Restrictions: only allowed in programs

Remarks: There can be up to 32 nested gosub statements in a program.

Example:

```
     DE1                    (define program 1)
     SAP:0                  (set axis position register)
     VL=1                   (load velocity)
     AC=10                  (load acceleration)
     RFN                    (run forward)
     PRT "Press any key to
           stop axis"       (print string to display with carriage return and line feed)
 1   IF KY GS5              (conditionally gosub 5)
     IF IP GT10             (conditionally goto 10)
     GT1                    (unconditionally goto 1)
 5   PNT "Axis position is "(print string to display without carriage return and line feed)
     FV1=AP                 (load floating point variable)
     OUT FV1                (output variable value to display)
     PRT " units."          (print string to display with carriage return and line feed)
     EKB                    (empty key buffer)
     ST                     (stop all motion)
     RET                    (return from gosub)
10   ED                     (end program 1 and exit program editor)
```

What will happen:

This program, once executed, runs the axis in the forward direction. It then waits for a character from the key buffer and goes to the subroutine at label 5 upon receiving the character. This subroutine prints the axis position on the display, empties the key buffer, stops the axis and returns. Lastly, once the axis is in position (IP), it goes to the statement at label 10, which ends the program.

See Also: GS, IF...GT, RET, POP, RST

IF...GT — CONDITIONALLY "GOTOS" LABEL

Type: program

Syntax: IF *p1* GT*p2*

Parameters:

	p1	*p2*
type	bexpr	integer **or** integer variable **or** pointed integer variable
minimum		1
maximum		999
allowed values	0, 1	

Usage: This command causes the program to conditionally go to label *p2* if *p1* is true (i.e., evaluates to 1).

Restrictions: only allowed in programs

Example:

```
     DE1                      (define program 1)
     SAP:0                    (set axis position register)
     VL=1                     (load velocity)
     AC=10                    (load acceleration)
     RFN                      (run forward)
     PRT "Press any key to
          stop axis"          (print string to display with carriage return and line feed)
1    IF KY GS5                (conditionally gosub 5)
     IF IP GT10               (conditionally goto 10)
     GT1                      (unconditionally goto 1)
5    PNT "Axis position is "  (print string to display without carriage return and line feed)
     FV1=AP                   (load floating point variable)
     OUT FV1                  (output variable value to display)
     PRT " units."            (print string to display with carriage return and line feed)
     EKB                      (empty key buffer)
     ST                       (stop all motion)
     RET                      (return from gosub)
10   ED                       (end program 1 and exit program editor)
```

What will happen:

This program, once executed, runs the axis in the forward direction. It then waits for a character from the key buffer and goes to the subroutine at label 5 upon receiving the character. This subroutine prints the axis position on the display, empties the key buffer, stops the axis and returns. Lastly, once the axis is in position (IP), it goes to the statement at label 10, which ends the program.

See Also: GT
IF...GS

IM — LOADS INCREMENTAL MOVE POSITION

Type: register

Syntax: IM=*p1*

Parameters:

	p1
type	vexpr
units	units
default	0 pulses
minimum	-2,000,000,000 pulses
maximum	2,000,000,000 pulses

Usage: This command loads the incremental move position register.

Remarks: The numerical values for the default, minimum and maximum of the parameter shown above are assuming that the axis unit ratio, AUR, is set at its default value of 1. If the axis unit ratio is set to a value other than 1, the default, maximum and minimum values will change appropriately (see AUR).

Example:

VL=10	(load velocity)
AC=40	(load acceleration)
IM=12	(load incremental move position)
RIN	(run to incremental move position)

What will happen:

By setting the speed, acceleration, incremental move position and issuing the RIN command, the axis will move 12 units in the forward direction.

See Also: RIN
AM
OM
AUR

IN — INPUTS VARIABLE VALUE FROM KEY BUFFER

Type: input/output

Syntax: IN *p1*

Parameters:

	p1
type	variable
allowed values	IV*n*, IVP*n*, FV*n*, FVP*n*, SV*n*, SVIV*n*, SVIVP*n*

Usage: This command inputs a variable value from the key buffer.

Restrictions: allowed in programs only

Remarks:

1. When using IV*n*, IVP*n*, FV*n*, or FVP*n*:
   a. if the number is greater than 30 characters long, or if it is out of the numerical range of the variable, then bit 5 in the program status register will be set to 1, which means 'String value out of range'. A zero will be loaded into the variable.
   b. if one or more of the characters are not valid, then bit 4 in the program status register will be set to 1, which means 'Invalid digit in string'. A zero will be loaded into the variable.
2. When using SV*n*, SVIV*n* or SVIVP*n*, if the string entered is greater than 127 characters, only the first 127 characters will be loaded. The rest will stay in the key buffer.

Examples:

```
    DE1                          (define program 1)
    PRT "Enter an integer:"      (print string to display with carriage return and line feed)
1   IN IV1                       (input variable value from key buffer)
    IF N SE1 GT2                 (conditionally goto 2)
    PRT "Invalid number -
         Enter again"            (print string to display with carriage return and line feed)
    GT1                          (unconditionally goto 1)
2   PRT "Enter a string:"        (print string to display with carriage return and line feed)
    IN SV1                       (input variable value from key buffer)
    ED                           (end program 1 and exit program editor)
```

What will happen:

This program, once executed, will prompt the user to enter an integer. After the user enters the number, the program checks to see if both program status register bits 4 and 5 (SE1) are not set. If either one is set, the program prints an error message and asks the user to enter it again. If neither one is set, the program goes to 2, where the user will be prompted to enter a string. Once it is entered, the program ends.

See Also: GET, OUT, RPS

IOP — INCREMENTS OFFSET POSITION REGISTER

Type: register

Syntax: IOP:*p1*

Parameters:

	p1
type	vexpr
units	units
default	0 pulses
minimum	-2,000,000,000 pulses
maximum	2,000,000,000 pulses

Usage: This command increments the offset position register.

Remarks: The numerical values for the default, minimum and maximum of the parameter *p1* shown above are assuming that the axis unit ratio, AUR, is set at its default value of 1. If the axis unit ratio is set to a value other than 1, the default, maximum and minimum values will change appropriately (see AUR).

Example:

SOP:0	(set offset position register)
VL=10	(load velocity)
AC=40	(load acceleration)
OM=10	(load offset move position)
RON	(run to offset move position)
IOP:-10	(increment offset position register)
RON	(run to offset move position)

What will happen:

By setting the offset position register to 0, loading the velocity, acceleration and offset move position and issuing the RON command, the axis will move 10 units in the forward direction. After the axis has stopped, the offset position register will be at 10 units. By issuing the IOP command, the offset position register will be incremented by -10 units (i.e., decremented by 10 units), and therefore will be set to 0. By again issuing the RON command, the axis will again move 10 units in the forward direction.

See Also: SOP
RON
AUR

IV — LOADS INTEGER VARIABLE

Type: variable

Syntax: IV*p1=p2*

Parameters:

	p1	*p2*
type	integer	vexpr
minimum	1	-2,147,483,648
maximum	4096	2,147,483,647

Usage: This command loads integer variable *p1* with the result of the evaluated variable expression *p2*.

Remarks: The numerical value for the maximum of the parameter *p1* shown above is assuming that the floating point variable allocation, FP, is set at its default value of 0. If the floating point variable allocation is set to a value other than 0, the maximum value will change appropriately (see FP).

Examples:

IV1=15	(load integer variable 1 with 15)
IV2=32	(load integer variable 2 with 32)
IV3=IV1 + IV2	(load integer variable 3 with 47)
IV4=SQR IV3* IV2	(load integer variable 4 with square root of 47*32)
IV5=RR*2	(load integer variable 5 with the resolver reading * 2)

See Also: DV
FP
RIV

IVP — LOADS INTEGER VARIABLE POINTED TO BY ANOTHER INTEGER VARIABLE

Type: variable

Syntax: IVP*p1=p2*

Parameters:

	p1	*p2*
type	integer	vexpr
minimum	1	-2,147,483,648
maximum	4096	2,147,483,647

Usage: This command loads the integer variable pointed to by integer variable *p1* with the result of the evaluated variable expression *p2*.

Remarks: The numerical value for the maximum of the parameter *p1* shown above is assuming that the floating point variable allocation, FP, is set at its default value of 0. If the floating point variable allocation is set to a value other than 0, the maximum value will change appropriately (see FP).

Example:

```
IV4=67      (load integer variable with pointer)
IVP4=43     (load integer variable)
```

What will happen:

Integer variable number 67, which is pointed to by integer variable 4, will be loaded with the value 43.

See Also: IV, FP, RIV

JK — LOADS JERK PERCENTAGE

Type: register

Syntax: JK=*p1*

Parameters: *p1*

type	vexpr
units	%
default	0
minimum	0
maximum	100

Usage: This command loads the jerk percentage. The jerk percentage is defined as the percentage of acceleration/deceleration time that the axis will jerk. JK is used when the motion parameter, MP, is assigned to either time or velocity.

Remarks: If JK is set to 0, there is no jerk limit, i.e. the jerk is infinite.

Example:

SAP:0	(set axis position register)
VL=5	(load velocity)
AC=10	(load acceleration)
IM=40	(load incremental move position)
JK=100	(load jerk percentage)
RIN	(run to incremental move position)
JK=0	(load jerk percentage)
RIN	(run to incremental move position)

What will happen:

By setting the axis position register, loading the velocity, acceleration, incremental move position, jerk percentage and issuing the RIN command, the axis will move 40 units in the forward direction. The axis will jerk for the whole time it is accelerating and decelerating. Then, by loading 0 into the jerk percentage and again issuing the RIN command, the axis will instantaneously achieve the acceleration rate and deceleration rate during the move.

KA — LOADS ACCELERATION FEEDFORWARD

Type: axis

Syntax: KA=*p1*

Parameters:

	p1
type	vexpr
default	0
minimum	0
maximum	64,000

Usage: This command loads the acceleration feedforward constant.

Restrictions: servo only

Remarks: The acceleration feedforward constant is used to reduce following error during acceleration or deceleration. The equation for setting KA based on the torque to inertia ratio and the axis feedback resolution, FR, is:

$$KA = \frac{2^{32}\pi}{FR} \times \frac{1}{\left(\frac{torque}{inertia}\right)}$$

This value along with the values of all the other control constants can be set automatically by the KS command.

See Also: FR
KS

KD — LOADS DERIVATIVE CONTROL GAIN

Type: axis

Syntax: KD=*p1*

Parameters:

	p1
type	vexpr
default	500
minimum	0
maximum	8,000

Usage: This command loads the derivative control gain.

Restrictions: servo only

Remarks: The derivative control gain is used to multiply the time derivative of the following error to control the position of the axis. The equation for setting KD based on the torque to inertia ratio and the axis feedback resolution, FR, is:

$$KD = \frac{316,022,860}{FR} \times \frac{1}{\sqrt{\frac{torque}{inertia}}}$$

This value along with the values of all the other control constants can be set automatically by the KS command.

See Also: FR
KS

KI — LOADS INTEGRAL CONTROL GAIN

Type: axis

Syntax: KI=*p1*

Parameters:

p1	
type	vexpr
default	0
minimum	0
maximum	64,000

Usage: This command loads the integral control gain.

Restrictions: servo only

Remarks: The integral control gain is used to multiply the time integral of the following error to control the position of the axis. The equation for setting KI based on the torque to inertia ratio and the axis feedback resolution, FR, is:

$$KI = \frac{686,310}{FR} \times \sqrt{\frac{torque}{inertia}}$$

This value along with the values of all the other control constants can be set automatically by the KS command.

See Also: FR
KS

KL — KILLS PROGRAM

Type: program

Syntax: KL*p1*

Parameters: *p1*

type	integer **or** integer variable **or** pointed integer variable
minimum	1
maximum	4

Usage: This command kills program *p1*, i.e. stops its execution.

Restrictions: not allowed in motion blocks

Remarks: This command will not stop any motion caused by program *p1*.

See Also: KLA

KLA — KILLS ALL PROGRAMS

Type: program

Syntax: KLA

Usage: This command kills all programs, i.e. stops their execution.

Restrictions: not allowed in motion blocks

Remarks: This command will not stop any motion caused by the programs previously executing.

See Also: KL

KP — LOADS PROPORTIONAL CONTROL GAIN

Type: axis

Syntax: KP=*p1*

Parameters:

p1	
type	vexpr
default	10
minimum	0
maximum	8,000

Usage: This command loads the proportional control gain.

Restrictions: servo only

Remarks: The proportional control gain is used to multiply the following error to control the position of the axis. The equation for setting KP based on the axis feedback resolution, FR, is:

$$KP = \frac{327,680}{FR}$$

This value along with the values of all the other control constants can be set automatically by the KS command.

See Also: FR
KS

KS AUTOMATICALLY SETS UP CONTROL CONSTANTS

Type: axis

Syntax: KS

Usage: This command automatically sets up the control constants, which are FT, KA, KD, KI, and KP.

Restrictions: servo only; not allowed in programs or motion blocks

Remarks: This command will execute only when the controller is faulted and no programs or motion blocks are executing. The motor should be connected to the load when using this command. When executed, it causes the axis to move 1000 pulses in the forward direction. Be sure that the axis is free to move this far before executing this command. This command takes about two seconds to execute, and when finished, the controller will return either an asterisk (*) indicating successful completion or a question mark (?) followed by the appropriate error message. The possible error messages are as follows:

1. TORQUE TO INERTIA RATIO TOO LOW -- the torque to inertia ratio of the axis is less than 125 radians/sec^2.
2. TORQUE TO INERTIA RATIO TOO HIGH -- the torque to inertia ratio of the axis is greater than 125,000 radians/sec^2.
3. TORQUE RESPONSE NON-LINEAR -- auto-tuning won't work.

See Also: FT
KA
KD
KI
KP

KY — PUTS ONE CHARACTER INTO KEY BUFFER

Type: system

Syntax: KY*p1*

Parameters:

	p1
type	ASCII character
minimum	ASCII character 0 (NULL)
maximum	ASCII character 127 (DEL)

Usage: This command puts one character into the key buffer.

Restrictions: not allowed in motion blocks

Example:

```
KYE     (put "E" into key buffer)
KY1     (put "1" into key buffer)
```

See Also: IN

L — MAKES LAST STATEMENT THE CURRENT STATEMENT IN THE PROGRAM/MOTION BLOCK EDITOR

Type: program

Syntax: L

Usage: This command makes the last statement the current statement in the program or motion block editor.

Restrictions: not allowed out of programs or motion blocks

Example:

```
DE1                 (define program 1)
*  SAP:0
X                   (step through program)
*  VL=10
X                   (step through program)
*  AC=40
L                   (make last statement the current statement)
*  VL=10
4!                  (exit program editor)
*
```

See Also: DE
MB
X

LB — MAKES STATEMENT AT LABEL THE CURRENT STATEMENT IN PROGRAM EDITOR

Type: program

Syntax: LB*p1*

Parameters: *p1*

type	integer
minimum	1
maximum	999

Usage: This command makes the statement at label *p1* the current statement in the program editor.

Restrictions: only allowed in programs

Example:

```
DE1                    (define program 1)
*   SAP:0
LB5                    (make statement at label 5 current statement)
*005PRT"Press any key to stop axis"
!                      (exit from program editor)
*
```

See Also: DE
L
X

LK — LOCKS INTERPRETER TO PROGRAM

Type: program

Syntax: LK

Usage: This command locks the interpreter to the program, which causes other currently executing programs to be to suspended.

Restrictions: only allowed in programs

Remarks: Once a program containing the LK command is done executing, the interpreter will automatically be unlocked from that program.

Example:

```
    DE1          (define program 1)
1   WT TM1       (wait for expression to be true)
    LK           (lock interpreter to program)
    IF KY GT2    (conditionally goto 2)
    ULK          (unlock interpreter from program)
    GT1          (unconditionally goto 1)
2   ED           (end program and exit program editor)

    STM1=0.01    (load start time of timer 1 and start timer 1)
```

What will happen:

This program, once executed, will first wait for 10 ms. Then, it locks the interpreter and checks for KY to be true, i.e. for a character to be entered into the key buffer. If KY is true, then the program goes to the statement at label 2, which ends the program. If it is not, then it unlocks the interpreter and goes to the statement at label 1, which waits for 10 ms, etc.

See Also: ULK

MB — DEFINES MOTION BLOCK

Type: program

Syntax: MB*p1*

Parameters:

p1	
type	integer
minimum	1
maximum	100

Usage: This command enters the motion block line editor at the first statement of motion block *p1*. It is used to either view motion blocks or to start editing them.

Restrictions: not allowed in programs or motion blocks

Remarks: This command will execute only when the axis is stopped and no programs or motion blocks are executing.

Example:

MB1	(define motion block 1)
VL=10	(load velocity)
AC=40	(load acceleration rate)
IM=15	(load incremental move position)
RIN	(run to incremental move position)
MED	(end motion block 1 and exit motion block editor)

See Also: DE, MED, X, !, DEL, L, FLT

MD — ASSIGNS DIRECTION OF MOTOR FOR FORWARD MOVES

Type: axis

Syntax: MD=*p1*

Parameters:
p1
default: CW
allowed values: CW, CC

Usage: This command assigns the direction of the motor for forward moves. If MD is set to CW, a forward move by the motor is clockwise, facing the motor shaft. If MD is set to CC, a forward move by the motor is counterclockwise, facing the motor shaft.

Restrictions: not allowed in programs or motion blocks

MED — ENDS MOTION BLOCK AND EXITS MOTION BLOCK EDITOR

Type: program

Syntax: MED

Usage: This command marks the end of a motion block and exits from the motion block editor.

Restrictions: only allowed in motion blocks

Remarks: Be cautious when using this command, since it will delete all motion block statements that follow it. If you only wish to exit the motion block editor, use the ! command.

Example:

```
MB1     (define motion block 1)
VL=10   (load velocity)
AC=40   (load acceleration rate)
IM=15   (load incremental move position)
RIN     (run to incremental move position)
MED     (end motion block 1 and exit motion block editor)
```

See Also: !

MEX — EXECUTES MOTION BLOCK

Type: program

Syntax: MEX*p1*

Parameters:

p1	
type	integer **or** integer variable **or** pointed integer variable
minimum	1
maximum	100

Usage: This command executes motion block *p1*.

Restrictions: not allowed in motion blocks

Remarks: If a motion block is executing, the MEX command will quit executing that motion block and then execute motion block *p1*. If motion block *p1* is already executing, then it is restarted by MEX*p1*.

Example:

```
MB1     (define motion block 1)
VL=10   (load velocity)
AC=40   (load acceleration rate)
IM=15   (load incremental move position)
RIN     (run to incremental move position)
MED     (end motion block 1 and exit motion block editor)

MEX1    (execute motion block 1)
```

What will happen:

After issuing the MEX command, the axis will move 15 units in the forward direction.

See Also: EX

MP — ASSIGNS MOTION PARAMETER

Type: register

Syntax: MP=*p1*

Parameters:

p1	
default	V
allowed values	V, T

Usage: This command assigns the motion parameter to either velocity (V) or time (T).

Remarks:

1. When MP is set to V, the motion commands make use of VL, AC, DC and JK to define the motion profile.
2. When MP is set to T, the motion commands make use of MVT, ACP, DCP, and JK to define the motion profile.

MRP — REPEATS MOTION FROM BEGINNING OF MOTION BLOCK

Type: program

Syntax: MRP

Usage: This command causes the motion block to repeat motion from the beginning of the motion block.

Restrictions: only allowed in motion blocks

Example:

```
MB1     (define motion block 1)
VL=10   (load velocity)
AC=40   (load acceleration rate)
IM=15   (load incremental move position)
AM=0    (load absolute move position)
RIN     (run to incremental move position)
RAN     (run to absolute move position)
MRP     (repeat motion from beginning of motion block)
MED     (end motion block 1 and exit motion block editor)
```

What will happen:

This motion block, when executed, will load the velocity, acceleration rate, incremental move position, and absolute move position. Next, the axis will move 15 units in the forward direction. Once the motion is completed, the axis will then move 15 units in the reverse direction. It will then repeat this motion until a motion command or another motion block is executed.

MS — AUTOMATICALLY SETS UP MOTOR CONSTANTS

Type: axis

Syntax: MS

Usage: This command automatically sets up the motor constants, which are CO and PR.

Restrictions: resolver feedback servo only; not allowed in programs or motion blocks

Remarks: This command will execute only when the controller is faulted and no programs or motion blocks are executing. The motor should not be connected to a load when using this command. When executed, it causes the motor rotor to line up with two locations of the stator vector. This command takes about two seconds to execute, and when finished, the controller will return either an asterisk (*) indicating successful completion or a question mark (?) followed by the appropriate error message. The possible error messages are as follows:

1. SWITCH MOTOR LEADS -- two motor leads should be switched.
2. BAD POLES RATIO -- the motor poles to resolver poles ratio was less than 1 or greater than 16.

See Also: CO
PR

MVT — LOADS MOVE TIME

Type: register

Syntax: MVT=*p1*

Parameters:

	p1
type	vexpr
units	seconds
default	650.00
minimum	.01
maximum	650.00

Usage: This command loads the move time. MVT is used when the motion parameter, MP, is assigned to time.

Example:

IM=5	(load incremental move position)
MP=T	(assign motion parameter to time)
MVT=10	(load move time)
ACP=40	(load acceleration percentage)
RIN	(run to incremental move position)

What will happen:

By setting the incremental move position, move time, acceleration percentage and issuing the RIN command, the axis will move 5 units in the forward direction in 10 seconds.

See Also: MP

OFE — ENABLES FAULT ON OUTPUT FAULT

Type: input/output

Syntax: OFE=*p1*

Parameters:

p1	
type	boolean **or** bexpr
default	0
allowed values	0, 1

Usage: This command is used to enable the controller to fault on a digital output fault. If OFE is set to 1, the fault on output fault is enabled, and if OFE is set to 0, it is disabled.

Restrictions: not allowed in motion blocks

OM — LOADS OFFSET MOVE POSITION

Type: register

Syntax: OM=*p1*

Parameters:

	p1
type	vexpr
units	units
default	0 pulses
minimum	-2,000,000,000 pulses
maximum	2,000,000,000 pulses

Usage: This command loads the offset move position register.

Remarks: The numerical values for the default, minimum and maximum of the parameter shown above are assuming that the axis unit ratio, AUR, is set at its default value of 1. If the axis unit ratio is set to a value other than 1, the default, maximum and minimum values will change appropriately (see AUR).

Example:

SOP:0	(set offset position register)
VL=10	(load velocity)
AC=40	(load acceleration rate)
OM=8	(load offset move position)
RON	(run to offset move position)

What will happen:

By setting the offset position register, velocity, acceleration, offset move position and issuing the RON command, the axis will move 8 units in the forward direction.

See Also: RON
AM
IM
AUR

OUT — OUTPUTS VARIABLE VALUE TO DISPLAY

Type: input/output

Syntax: OUT *p1*

Parameters:

p1	
type	variable
allowed values	IV*n*, IVP*n*, FV*n*, FVP*n*, SV*n*, SVIV*n*, SVIVP*n*

Usage: This command outputs a variable value to the display. If it is a number variable (i.e., IV*n*, IVP*n*, FV*n* or FVP*n*), the number will be formatted according to the field width, FW, and the number of decimal places, DP.

Remarks:

1. If the number does not fit into the specified field width, FW, or number of decimal places, DP, then asterisks will be sent to the display.
2. If display format is disabled, the variable value will be sent to the terminal.

Examples:

```
FW=6                    (assign field width of output variable on display)
IV1=50                  (load integer variable)
OUT IV1                 (output variable value to display)
*      50
FV1=50.2                (load floating point variable)
DP=2                    (assign number of decimal places after decimal point on display)
OUT FV1                 (output variable value to display)
* 50.20
SV1="TEST"              (load string variable)
OUT SV1                 (output variable value to display)
*TEST
```

See Also: DSE, DP, FW, PRT, PNT

PB — LOADS IN-POSITION BAND

Type: axis

Syntax: PB=*p1*

Parameters:

	p1
type	vexpr
units	units
default	0 pulses
minimum	0 pulses
maximum	16,000 pulses

Usage: This command loads the in-position band of the axis. This is the maximum amount of position error that the axis can have while it is in position.

Remarks: The numerical values for the default, minimum and maximum of the parameter *p1* shown above are assuming that the axis unit ratio, AUR, is set at its default value of 1. If the axis unit ratio is set to a value other than 1, the default, maximum and minimum values will change appropriately (see AUR).

PC — LOADS PEAK CURRENT IN PERCENT OF MAXIMUM

Type: axis

Syntax: PC=*p1*

Parameters:

p1	
type	vexpr
units	%
default	100
minimum	1
maximum	100

Usage: This command loads the peak current available for the motor in percent of the peak current rating of the drive.

Restrictions: servo only

See Also: CC
SC
TC

PF — ASSIGNS POSITION FEEDBACK INPUT TYPE

Type: axis

Syntax: PF=*p1*

Parameters:
p1
default: Q4
allowed values: PD, Q1, Q2, Q4

Usage: This command is used to assign the type of position feedback. The possibilities are listed below:

PD - pulse/direction. This sets the feedback for a pulse input on channel A and a direction input on channel B.

Q1 - quadrature x1. This sets the feedback for two pulse waveforms in quadrature with a pulse multiplier of 1.

Q2 - quadrature x2. This sets the feedback for two pulse waveforms in quadrature with a pulse multiplier of 2.

Q4 - quadrature x4. This sets the feedback for two pulse waveforms in quadrature with a pulse multiplier of 4.

Restrictions: stepper and encoder feedback servo only; not allowed in programs or motion blocks

See Also: FR
PFE

PFD — LOADS POSITION FEEDBACK DENOMINATOR

Type: axis

Syntax: PFD=*p1*

Parameters:

p1

type	integer
default	1 (servo) **or** 4 (stepper)
minimum	1
maximum	10,000

Usage: This command loads the position feedback denominator. It is a parameter used in dual loop feedback for servos, and in encoder position feedback for steppers. It is defined as the denominator of the position feedback ratio between the axis and the auxiliary encoder input for servos, and the denominator of the ratio between 50,000 and the encoder position feedback for steppers. For example, if you had a stepper with an encoder of 1,000 pulses/revolution and the position feedback type is Q4 (quadrature x4), your ratio would be 50,000/(1,000*4) = 50,000/4,000 = 50/4. Therefore, you would set PFN equal to 50 and PFD equal to 4.

Restrictions: stepper or dual loop feedback servo only

See Also: PFN
PFE
XFE

PFE — ENABLES POSITION FEEDBACK

Type: axis

Syntax: PFE=*p1*

Parameters:

p1	
type	boolean
default	0
allowed values	0, 1

Usage: This command is used to enable encoder position feedback for stepper motors. If PFE is set to 1, then position feedback is enabled, and if PFE is set to 0, it is disabled.

Restrictions: stepper only; not allowed in programs or motion blocks

Remarks: When position feedback is enabled, the axis position, AP, is set to the auxiliary encoder input. When position feedback is disabled, the axis position, AP, is set to the command position, CP.

See Also: PF
BL
CT
DB

PFN — LOADS POSITION FEEDBACK NUMERATOR

Type: axis

Syntax: PFN=*p1*

Parameters:

p1

type	integer
default	0 (servo) **or** 50 (stepper)
minimum	0
maximum	10,000

Usage: This command loads the position feedback numerator. It is a parameter used in dual loop feedback for servos, and in encoder position feedback for steppers. It is defined as the numerator of the position feedback ratio between the axis and the auxiliary encoder input for servos, and the numerator of the ratio between 50,000 and the encoder position feedback for steppers. For example, if you had a stepper with an encoder of 1,000 pulses/revolution and the position feedback type is Q4 (quadrature x4), your ratio would be 50,000/(1,000*4) = 50,000/4,000 = 50/4. Therefore, you would set PFN equal to 50 and PFD equal to 4.

Restrictions: stepper or dual loop feedback servo only

See Also: PFD
PFE
XFE

PHB — LOADS PHASE ERROR BOUND

Type: register

Syntax: PHB=*p1*

Parameters:

p1	
type	vexpr
units	pulses
default	32,000
minimum	0
maximum	32,000

Usage: The phase error bound is used to define a bound on the phase error of the phase locked loop. If this limit is exceeded, the phase error is set to half of the phase error bound and bit five of the axis status register is set to 1. This corresponds to the axis status message "Phase error past bound".

Restrictions: extended command set only

See Also: RPE
PHE

PHE — ENABLES PHASE LOCKED LOOP

Type: motion

Syntax: PHE=*p1*

Parameters:

p1	
type	bexpr
default	0
allowed values	0, 1

Usage: This command is used to enable the phase locked loop. If PHE is set to 1, then the phase locked loop is enabled, and if PHE is set to 0, it is disabled.

Restrictions: extended command set only

See Also:
PHL
PHO
PHG
PHZ
PHB
PHT
RPE
RPM
RPP

PHG — LOADS PHASE GAIN

Type: register

Syntax: PHG=*p1*

Parameters:

	p1
type	vexpr
default	0
minimum	0
maximum	255

Usage: This command loads the phase gain of the phase locked loop. The phase gain is used to multiply the phase error to adjust the value of the phase multiplier.

Restrictions: extended command set only

See Also: RPP
RPM
PHE

PHL — LOADS PHASE LENGTH

Type: register

Syntax: PHL=*p1*

Parameters:

p1	
type	vexpr
units	pulses
default	1000
minimum	500
maximum	64,000

Usage: This command loads the phase length of the phase locked loop. The phase length is used to define the number of pulses during one cycle of the reference input.

Restrictions: extended command set only

See Also: RPP
SPH

PHO — LOADS PHASE OFFSET

Type: register

Syntax: PHO=*p1*

Parameters:

	p1
type	vexpr
units	pulses
default	0
minimum	-32,000
maximum	32,000

Usage: This command loads the phase offset, which is an offset on the reference position of the phase locked loop.

Restrictions: extended command set only

See Also: RPP
PHE

PHT — LOADS PHASE LOCKOUT TIME

Type: register

Syntax: PHT=*p1*

Parameters: *p1*

type	vexpr
units	seconds
default	0.05
minimum	.001
maximum	4.000

Usage: This command is used to load the phase lockout time of the phase locked loop. The phase lockout time is the time interval after the position capture in which the phase locked loop is disabled. This time interval is used to account for any undesired position capture inputs.

Restrictions: extended command set only

See Also: PHE

PHZ — LOADS PHASE ZERO

Type: register

Syntax: PHZ=*p1*

Parameters:

p1	
type	vexpr
default	245
minimum	0
maximum	255

Usage: This command is used to load the the zero of the compensator of the phase locked loop. This, is conjunction with PHG, defines a method of correction of the phase in the phase locked loop.

Restrictions: extended command set only

See Also: PHG
PHE

PNT — PRINTS STRING TO DISPLAY WITHOUT CARRIAGE RETURN AND LINE FEED

Type: input/output

Syntax: PNT"*p1*"

Parameters:

	p1
type	string
minimum	0 characters
maximum	127 characters

Usage: This command prints string *p1* to the display without a carriage return or line feed.

Remarks: If display format is disabled, the string will be sent to the terminal.

Example:

```
DE1                 (define program 1)
PNT"Testing"        (print string to display without carriage return and line feed)
PNT" PNT command"   (print string to display without carriage return and line feed)
ED                  (end program 1 and exit program editor)

EX1                 (execute program 1)
*Testing PNT command
```

See Also: DSE
PRT
OUT

POE — ENABLES POWER OUTPUT STAGE

Type: axis

Syntax: POE=*p1*

Parameters: *p1*

type	boolean **or** bexpr
default	1
allowed values	0, 1

Usage: This command is used to enable the output stage of the amplifier. If POE is set to 1, then the power output stage is enabled, and if POE is set to 0, it is disabled.

POP — POPS "GOSUB" ADDRESS FROM TOP OF "GOSUB" STACK

Type: program

Syntax: POP

Usage: This command pops the last gosub address from the top of the gosub stack. It causes the program to exit a subroutine without returning.

Restrictions: only allowed in programs

Example:

```
    DE1                     (define program 1)
    VL=5                    (load velocity)
    AC=40                   (load acceleration)
    AM=10                   (load absolute move position)
    GS10                    (unconditionally gosub 10)
    GT20                    (unconditionally goto 20)
10  RAN                     (run to absolute move position)
11  IF IP GT12              (conditionally goto 12)
    IV1=FC                  (load integer variable)
    IF IV1 <> 0 GT15        (conditionally goto 15)
    GT11                    (unconditionally goto 11)
12  RET                     (return from gosub)
15  POP                     (pop gosub address from top of gosub stack)
    PRT"CONTROLLER
            FAULT"          (print string to display with carriage return and line feed)
    PRT"TYPE 'RFC' FOR
            EXPLANATION"
                            (print string to display with carriage return and line feed)
20  ED                      (end program 1 and exit program editor)
```

What will happen:

This program, when executed, will load the velocity, acceleration rate and absolute move position. It will then goto the subroutine at label 10, which will run the axis in the forward direction for 10 units. While the axis is running, the program checks two things: 1) to see if the axis is in position (IP) and 2) to see if a fault has occurred (FC<>0). If a fault has occurred, the program will goto label 15. Then, the program will pop the address of label 10 off of the stack, print out an error message and end. If a fault does not occur, the program will return to the statement after "GS10", which goes to the statement at label 20, which ends the program.

See Also: GS
RET
RST

PR — LOADS MOTOR POLES TO RESOLVER POLES RATIO

Type: axis

Syntax: PR=*p1*

Parameters:

p1	
type	vexpr
default	2
minimum	1
maximum	16

Usage: This command loads the motor poles to resolver poles ratio of the motor.

Restrictions: resolver feedback servo only

Remarks: This value along with the value of CO can be set automatically by the MS command.

See Also: MS

PRT — PRINTS STRING TO DISPLAY WITH CARRIAGE RETURN AND LINE FEED

Type: input/output

Syntax: PRT"*p1*"

Parameters:

p1	
type	string
minimum	0 characters
maximum	127 characters

Usage: This command prints string *p1* to the display with a carriage return and line feed.

Remarks: If display format is disabled, the string will be sent to the terminal.

Example:

```
DE1                    (define program 1)
PRT"Testing"           (print string to display with carriage return and line feed)
PRT" PRT command"      (print string to display with carriage return and line feed)
ED                     (end program 1 and exit program editor)

EX1                    (execute program 1)
*Testing
PRT command
```

See Also: DSE, PNT, OUT

PW — PROMPTS FOR PASSWORD

Type: system

Syntax: PW

Usage: This command is used to prompt the user for the password.

Restrictions: not allowed in programs or motion blocks

Remarks: The password should be entered once the "Enter password:" prompt appears. The password can be up to 10 characters long. If the PW command has not been issued and the correct password not entered after power up, only diagnostic commands can be entered.

See Also: PWC

PWC — PROMPTS FOR PASSWORD CHANGE

Type: system

Syntax: PWC

Usage: This command is used to prompt the user for a password change.

Restrictions: not allowed in programs or motion blocks

Remarks: Once the PWC command has been issued, the controller will prompt for the old password. After the old password has been entered, the controller will then prompt for the new password. The password can be up to 10 characters long. After the new password has been entered, the controller will prompt for the new password again for verification. Once this has been entered, the password will be changed to the new value.

See Also: PW

PWE — ENABLES POSITION REGISTER WRAP

Type: axis

Syntax: PWE=*p1*

Parameters:

p1	
type	boolean
default	0
allowed values	0, 1

Usage: This command is used to enable position register wrap. If PWE is set to 1, position register wrap is enabled, and if PWE is set to 0, it is disabled.

Restrictions: not allowed in programs or motion blocks

Remarks: When position register wrap is enabled, the controller will use the axis position length value, APL, as a means of position register wraparound. The axis position register will count from -APL units to APL-1 units.

See Also: APL

RAI — REPORTS ANALOG INPUT

Type: diagnostic

Syntax: RAI

Usage: This command reports the value of the analog input.

Remarks: When the RAI command is executed, the value of the analog input will be given in volts.

Restrictions: extended command set only; not allowed in programs or motion blocks

See Also: ADB
AOF
RAO

RAN — RUNS TO ABSOLUTE MOVE POSITION

Type: motion

Syntax: RAN

Usage: This command runs the axis to the absolute move position, AM.

Remarks: The run commands RAN, RIN, RON, RFN, and RRN override each other unless they are used in a motion block.

Example:

```
SAP:0      (set axis position register)
VL=10      (load velocity)
AC=40      (load acceleration rate)
AM=8       (load absolute move position)
RAN        (run to absolute move position)
```

What will happen:

By setting the axis position register, velocity, acceleration, absolute move position and issuing the RAN command, the axis will move 8 units in the forward direction.

See Also: AM
RIN
RON
RFN
RRN

RAO — REPORTS ANALOG OUTPUT

Type: diagnostic

Syntax: RAO

Usage: This command reports the value of the analog output.

Restrictions: not allowed in programs or motion blocks

Remarks: When the RAO command is executed, the analog output will be given in volts.

See Also: AO

RAP — REPORTS AXIS POSITION

Type: diagnostic

Syntax: RAP

Usage: This command reports the axis position.

Restrictions: not allowed in programs or motion blocks

Remarks: When the RAP command is executed, the axis position will be given in units.

See Also: AUR

RAT — REPORTS AXIS POSITION CAPTURE

Type: diagnostic

Syntax: RAT

Usage: This command reports the axis position capture. This is the position captured when the position capture is used to capture the position of the axis.

Restrictions: not allowed in programs or motion blocks

Remarks: If a position has not been captured, then the axis position capture will be 0. After a position has been captured, the position can be reported using the RAT command, and then the axis position capture will be set to 0 until a position is captured again. The axis position capture will be given in units.

RAV — REPORTS AXIS VELOCITY

Type: diagnostic

Syntax: RAV

Usage: This command reports the axis velocity.

Restrictions: not allowed in programs or motion blocks

Remarks: When the RAV command is executed, the axis velocity will be given in units/sec.

See Also: AUR

RBV — REPORTS BOOLEAN VARIABLE VALUE

Type: diagnostic

Syntax: RBV*p1*

Parameters: *p1*

type	integer
minimum	1
maximum	256

Usage: This command reports the value of boolean variable *p1*.

Restrictions: not allowed in programs or motion blocks

Remarks: When the RBV command is executed, the boolean variable value will be given as a boolean number.

See Also: BV

RCP — REPORTS COMMAND POSITION

Type: diagnostic

Syntax: RCP

Usage: This command reports the command position of the axis.

Restrictions: not allowed in programs or motion blocks

Remarks: When the RCP command is executed, the command position will be given in units.

See Also: AUR

RD — LOADS RATIO DENOMINATOR

Type: register

Syntax: RD=*p1*

Parameters:

p1	
type	vexpr
default	1
minimum	1
maximum	10,000

Usage: This command is used to load the ratio denominator, which is a parameter used in electronic gearing. It is defined as the denominator of the ratio between the axis and the auxiliary encoder input.

Restrictions: extended command set only

See Also: RN
RIE

RDI — REPORTS DIGITAL INPUT

Type: diagnostic

Syntax: RDI*p1*

Parameters:

p1	
type	integer
minimum	1
maximum	12

Usage: This command reports the value of digital input *p1*.

Restrictions: not allowed in programs or motion blocks

Remarks:

1. When the RDI command is executed, the value of the digital input will be given as a boolean number.
2. Note that *p1* is optional. If it is not given, then digital inputs 1 through 12 will be reported as a 12-bit binary number. The left-most bit of the 12-bit binary number represents input 12, and the right-most bit represents input 1.

See Also: REG

RDO — REPORTS DIGITAL OUTPUT

Type: diagnostic

Syntax: RDO*p1*

Parameters:

p1	
type	integer
minimum	7
maximum	12

Usage: This command reports the value of digital output *p1*.

Restrictions: not allowed in programs or motion blocks

Remarks:

1. When the RDO command is executed, the value of the digital output will be given as a boolean number.
2. Note that *p1* is optional. If it is not given, then digital outputs 7 through 12 will be reported as a 6-bit binary number. The left-most bit of the 6-bit binary number represents output 12, and the right-most bit represents output 7.

REG — REPORTS POSITIVE EDGE SENSITIVE DIGITAL INPUT

Type: diagnostic

Syntax: REG*p1*

Parameters:

p1	
type	integer
minimum	1
maximum	12

Usage: This command reports the value of the positive edge sensitive digital input *p1*.

Restrictions: not allowed in programs or motion blocks

Remarks:

1. When the REG command is executed, the value of the positive edge sensitive digital input will be given as a boolean number. After the value has been reported using the REG command, it will be set to 0.
2. Note that *p1* is optional. If it is not given, then digital inputs 1 through 12 will be reported as a 12-bit binary number. The left-most bit of the 12-bit binary number represents input 12, and the right-most bit represents input 1.

See Also: RDI

RET — RETURNS FROM "GOSUB"

Type: program

Syntax: RET

Usage: This command causes the program to return from a subroutine to the statement after the gosub statement.

Restrictions: only allowed in programs

Example:

```
    DE1                  (define program 1)
    VL=5                 (load velocity)
    AC=40                (load acceleration)
    AM=10                (load absolute move position)
    GS10                 (unconditionally gosub 10)
    GT20                 (unconditionally goto 20)
10  RAN                  (run to absolute move position)
    WT IP                (wait for expression to be true)
    PRT"Axis in position" (print string to display with carriage return and line feed)
    RET                  (return from gosub)
20  ED                   (end program 1 and exit program editor)
```

What will happen:

This program, when executed, will load the velocity, acceleration rate and absolute move position. It will then goto the subroutine at label 10, which will run the axis in the forward direction for 10 units. Once the axis is in position, the program will print a string. Then, the program will return to the statement after "GS10", which goes to the statement at label 20, which ends the program.

See Also: GS
POP
RST

REV — REPORTS SOFTWARE REVISION

Type: diagnostic

Syntax: REV

Usage: This command reports the software revision of the controller software.

Restrictions: not allowed in programs or motion blocks

RFC — REPORTS FAULT CODE

Type: diagnostic

Syntax: RFC

Usage: This command reports the fault code.

Restrictions: not allowed in programs or motion blocks

Remarks:

1. When the RFC command is executed, the fault code will be given as an English statement that says which controller faults have occurred. If no fault has occurred, the controller returns "Controller functional".
2. If the computer interface format is enabled, and the RFC command is executed, the fault code will be given as an integer number. This number is the result of adding together all the powers of two associated with each fault code register bit equal to 1. If no fault has occurred, the fault code register is set to 0. The table below lists the possibilities:

number	bit	message
1	0	Power Failure
2	1	Reserved
4	2	Software Fault
8	3	Lost Enable
16	4	Digital Output Fault
32	5	Invalid Command
64	6	Command Not Allowed
128	7	Reserved
256	8	Mismatched Operands and Operator
512	9	Mathematical Overflow
1024	10	Mathematical Data Error
2048	11	Value Out of Range
4096	12	String Too Long
8192	13	Nonexistent Label
16384	14	Gosub Stack Underflow
32768	15	Gosub Stack Overflow
65536	16	Invalid Variable Pointer
131072	17	Invalid Motion
262144	18	Reserved
524288	19	Reserved
1048576	20	Reserved
2097152	21	Excessive Following Error
4194304	22	Excessive Command Increment
8388608	23	Position Register Overflow
16777216	24	Resolver Feedback Lost
33554432	25	Motor Power Over-Voltage
67108864	26	Motor Power Clamp Excessive Duty Cycle - Under-Voltage
134217728	27	Motor Power Clamp Over-Current Fault
268435456	28	Motor Over-Current Fault
536870912	29	Motor Over-Temperature
1073741824	30	Controller Over-Temperature
2147483648	31	Reserved

RFE — REPORTS FOLLOWING ERROR

Type: diagnostic

Syntax: RFE

Usage: This command reports the following error, which is the difference between the axis position, AP, and the command position, CP.

Restrictions: not allowed in programs or motion blocks

Remarks: When the RFE command is executed, the following error will be given in units.

See Also: AUR

RFI — REPORTS FAULT INPUT REGISTER

Type: diagnostic

Syntax: RFI

Usage: This command reports the fault input register.

Restrictions: not allowed in programs or motion blocks

Remarks:

1. When the RFI command is executed, the fault input register will be given as an English statement that says what fault inputs are active, if any. If none of the fault inputs are active, the controller returns "No fault input is active".
2. If the computer interface format is enabled, and the RFI command is executed, the fault input register will be given as an integer number. This number is the result of adding together all the powers of two associated with each fault input register bit equal to 1. If none of the fault inputs are active, the fault input register is set to 0. The table below lists the possibilities:

number	bit	message
1	0	Resolver feedback lost input active
2	1	Motor power over-voltage input active
4	2	Motor power clamp or under-voltage input active
8	3	Motor power clamp over-current input active
16	4	Motor over-current input active
32	5	Motor over-temperature input active
64	6	Controller over-temperature input active
128	7	Reserved
256	8	Reserved
512	9	Reserved
1024	10	Reserved
2048	11	Reserved
4096	12	Reserved
8192	13	Reserved
16384	14	Reserved
32768	15	Reserved

RFN — RUNS FORWARD

Type: motion

Syntax: RFN

Usage: This command runs the axis in the forward direction.

Remarks: The run commands RAN, RIN, RON, RFN, and RRN override each other unless they are used in a motion block.

Example:

```
VL=10        (load velocity)
AC=50        (load acceleration)
RFN          (run forward)
```

What will happen:

By loading the velocity and acceleration and issuing the RFN command, the axis will run in the forward direction until another motion command is issued.

See Also: RRN
RAN
RIN
RON

RFV — REPORTS FLOATING POINT VARIABLE VALUE

Type: diagnostic

Syntax: RFV*p1*

Parameters: *p1*

type	integer
minimum	1
maximum	2048

Usage: This command reports the value of floating point variable *p1*.

Restrictions: not allowed in programs or motion blocks

See Also: FV

RIE — ENABLES RATIO INPUT

Type: motion

Syntax: RIE=*p1*

Parameters:

	p1
type	bexpr
default	0
allowed values	0, 1

Usage: This command is used to enable the ratio input for electronic gearing. If RIE is set to 1, then the ratio input is enabled, and if RIE is set to 0, it is disabled.

Restrictions: extended command set only

See Also: RD
RN

RIN — RUNS TO INCREMENTAL MOVE POSITION

Type: motion

Syntax: RIN

Usage: This command runs the axis to the incremental move position, IM, i.e., it runs from the current position of the axis to the current position incremented by the value of IM.

Remarks: The run commands RAN, RIN, RON, RFN, and RRN override each other unless they are used in a motion block.

Example:

```
VL=10      (load velocity)
AC=40      (load acceleration rate)
IM=12      (load incremental move position)
RIN        (run to incremental move position)
```

What will happen:

By setting the velocity, acceleration, incremental move position and issuing the RIN command, the axis will move 12 units in the forward direction.

See Also: IM
RAN
RON
RFN
RRN

RIO — REPORTS I/O REGISTER

Type: diagnostic

Syntax: RIO

Usage: This command reports the I/O register.

Restrictions: not allowed in programs or motion blocks

Remarks:

1. When the RIO command is executed, the I/O register will be given as an English statement that says what inputs are active, if any. If none of the inputs are active, the controller returns "No I/O is active".
2. If the computer interface format is enabled, and the RIO command is executed, the I/O register will be given as an integer number. This number is the result of adding together all the powers of two associated with each I/O register bit equal to 1. If none of the inputs are active, the I/O register is set to 0. The table below lists the possibilities:

number	bit	message
1	0	Reserved
2	1	Reserved
4	2	Axis channel A input active
8	3	Axis channel B input active
16	4	Auxiliary channel A input active
32	5	Auxiliary channel B input active
64	6	Auxiliary index input active
128	7	Marker input active
256	8	Home input active
512	9	Forward overtravel input active
1024	10	Reverse overtravel input active
2048	11	Enable input active
4096	12	Capture input active
8192	13	Capture input edge
16384	14	Reserved
32768	15	OK output active

RIV — REPORTS INTEGER VARIABLE VALUE

Type: diagnostic

Syntax: RIV*p1*

Parameters:

p1	
type	integer
minimum	1
maximum	4096

Usage: This command reports the value of integer variable *p1*.

Restrictions: not allowed in programs or motion blocks

See Also: IV

RMR — REPORTS MEMORY REMAINING

Type: diagnostic

Syntax: RMR

Usage: This command reports the memory for programs and motion blocks remaining in the controller.

Restrictions: not allowed in programs or motion blocks

Remarks: When the RMR command is executed, the memory remaining will be given in bytes.

RN — LOADS RATIO NUMERATOR

Type: register

Syntax: RN=*p1*

Parameters:

p1	
type	vexpr
default	1
minimum	-10,000
maximum	10,000

Usage: This command is used to load the ratio numerator, which is a parameter used in electronic gearing. It is defined as the numerator of the ratio between the axis and the auxiliary encoder input.

Restrictions: extended command set only

See Also: RN
RIE

RO — LOADS REVERSE SOFTWARE OVERTRAVEL LIMIT

Type: axis

Syntax: RO=*p1*

Parameters:

	p1
type	vexpr
units	units
default	2,000,000,000 pulses
minimum	-2,000,000,000 pulses
maximum	2,000,000,000 pulses

Usage: This command loads the reverse software overtravel limit.

Remarks: The numerical values for the default, minimum and maximum of the parameter *p1* shown above are assuming that the axis unit ratio, AUR, is set at its default value of 1. If the axis unit ratio is set to a value other than 1, the default, maximum and minimum values will change appropriately (see AUR).

Example:

SAP:0	(set axis position register)
VL=10	(load velocity)
AC=40	(load acceleration)
AM=-15	(load absolute move position)
RO=-12	(load reverse software overtravel limit)
RAN	(run to absolute move position)

What will happen:

By setting the axis position register, velocity, acceleration, absolute move position, forward software overtravel and issuing the RAN command, the axis will move 12 units in the reverse direction and immediately halt all motion.

See Also: FO
AUR

RON — RUNS TO OFFSET MOVE POSITION

Type: motion

Syntax: RON

Usage: This command runs the axis to the offset move position.

Remarks: The run commands RAN, RIN, RON, RFN, and RRN override each other unless they are used in a motion block.

Example:

```
SOP:0      (set offset position register)
VL=10      (load velocity)
AC=40      (load acceleration rate)
OM=8       (load offset move position)
RON        (run to offset move position)
```

What will happen:

By setting the offset position register, velocity, acceleration, offset move position and issuing the RON command, the axis will move 8 units in the forward direction.

See Also: OM
RAN
RIN
RFN
RRN

RPE — REPORTS AXIS PHASE ERROR

Type: diagnostic

Syntax: RPE

Usage: This command is used to report the axis phase error of the phase locked loop. The phase error is the difference between the desired reference position and the reference position that was captured when the position capture input became active.

Restrictions: not allowed in programs or motion blocks; extended command set only

Remarks: When the RPE command is executed, the axis phase error will be given in pulses.

See Also: PHE

RPM — REPORTS AXIS PHASE MULTIPLIER

Type: diagnostic

Syntax: RPM

Usage: This command reports the axis phase multiplier, which is the ratio between the reference input and the axis output when using the phase locked loop.

Restrictions: not allowed in programs or motion blocks; extended command set only

See Also: PHE

RPP — REPORTS AXIS PHASE POSITION

Type: diagnostic

Syntax: RPP

Usage: This command is used to report the axis phase position. The phase position is the reference position of the phase locked loop.

Restrictions: not allowed in programs or motion blocks; extended command set only

Remarks: When the RPP command is executed, the axis phase position will be given in pulses.

See Also: SPH
PHE

RPS — REPORTS PROGRAM STATUS

Type: diagnostic

Syntax: RPS*p1*

Parameters:

	p1
type	integer
minimum	1
maximum	4

Usage: This command reports the status of program *p1*.

Restrictions: not allowed in programs or motion blocks

Remarks:

1. When the RPS command is executed, the program status will be given as an English statement. If no program is executing, the controller returns "Program not executing".
2. If the computer interface format is enabled, and the RPS command is executed, the program status will be given as an integer number. This number is the result of adding together all the powers of two associated with each program status register bit equal to 1. Note that if no program is executing, bit 0 will be set to 0, and the associated message is "Program not executing". The table below lists the possibilities:

number	bit	message
1	0	Program executing
2	1	Program locked out
4	2	Reserved
8	3	Reserved
16	4	Invalid digit in string
32	5	String value out of range
64	6	Reserved
128	7	Reserved
256	8	Reserved
512	9	Reserved
1024	10	Reserved
2048	11	Reserved
4096	12	Reserved
8192	13	Reserved
16384	14	Reserved
32768	15	PROGRAM FAULT

RRN — RUNS REVERSE

Type: motion

Syntax: RRN

Usage: This command runs the axis in the reverse direction.

Remarks: The run commands RAN, RIN, RON, RFN, and RRN override each other unless they are used in a motion block.

Example:

```
VL=10      (load velocity)
AC=50      (load acceleration)
RRN        (run reverse)
```

What will happen:

By loading the velocity and acceleration and issuing the RRN command, the axis will run in the reverse direction until another motion command is issued.

See Also: RFN
RAN
RIN
RON

RRR — REPORTS RESOLVER READING

Type: diagnostic

Syntax: RRR

Usage: This command reports the resolver reading.

Restrictions: resolver feedback servo only; not allowed in programs or motion blocks

RSB — RESETS BOOLEAN VARIABLE

Type: variable

Syntax: RSB*p1*

Parameters: *p1*

type	integer **or** integer variable **or** pointed integer variable
minimum	1
maximum	256

Usage: This command resets boolean variable *p1*, i.e. sets boolean variable *p1* to 0.

See Also: STB
BV
RBV

RSD — RESETS DIGITAL OUTPUT

Type: input/output

Syntax: RSD*p1*

Parameters:

p1	
type	integer
minimum	7
maximum	12

Usage: This command resets digital output *p1*, i.e. sets it to 0.

See Also: DOE, RSDA, STD, DO, RDO

RSDA — RESETS ALL DIGITAL OUTPUTS

Type: input/output

Syntax: RSDA

Usage: This command resets all digital outputs, i.e., sets them all to 0.

See Also: DOE
STDA
RSD
DO
RDO

RSF — RESETS FAULTS

Type: system

Syntax: RSF

Usage: This command resets all controller faults.

Restrictions: not allowed in motion blocks

See Also: RFC
STF

RSM	**RESUMES MOTION**
Type:	program
Syntax:	RSM
Usage:	This command resumes motion that has been suspended by the SUP command.
Restrictions:	not allowed in motion blocks
See Also:	SUP

RSRA — REPORTS AXIS STATUS REGISTER

Type: diagnostic

Syntax: RSRA

Usage: This command reports the axis status register.

Restrictions: not allowed in programs or motion blocks

Remarks:

1. When the RSRA command is executed, the axis status register value will be given as an English statement.
2. If the computer interface format is enabled, and the RSRA command is executed, the axis status register value will be given as an integer number. This number is the result of adding together all the powers of two associated with each axis status register bit equal to 1. Note that if the axis direction is reverse, bit 7 will be set to 0, and the associated message is "Axis direction reverse". The table below lists the possibilities:

number	bit	message
1	0	Motion generator enabled
2	1	Ratio input enabled
4	2	Phase locked loop enabled
8	3	Motion block executing
16	4	Motion suspended
32	5	Phase error past bound
64	6	Axis accel/decel
128	7	Axis direction forward
256	8	Axis in position
512	9	Axis at torque limit
1024	10	Axis at overtravel
2048	11	Axis at software overtravel
4096	12	Reserved
8192	13	AXIS FAULT
16384	14	Reserved
32768	15	Reserved

RSRS — REPORTS SYSTEM STATUS REGISTER

Type: diagnostic

Syntax: RSRS

Usage: This command reports the system status register.

Restrictions: not allowed in programs or motion blocks

Remarks:

1. When the RSRS command is executed, the system status register value will be given as an English statement.
2. If the computer interface format is enabled, and the RSRS command is executed, the system status register value will be given as an integer number. This number is the result of adding together all the powers of two associated with each system status register bit equal to 1. Note that if no program is executing, bit 0 will be set to 0, and the associated message is "No program executing". The table below lists the possibilities:

number	bit	message
1	0	Program executing
2	1	Program locked out
4	2	Reserved
8	3	Motion block executing
16	4	Key buffer empty
32	5	Transmit buffer full
64	6	Reserved
128	7	Reserved
256	8	Reserved
512	9	Reserved
1024	10	Reserved
2048	11	Reserved
4096	12	I/O FAULT
8192	13	AXIS FAULT
16384	14	SYSTEM FAULT
32768	15	MEMORY FAULT

RST — RESETS "GOSUB" STACK TO EMPTY

Type: program

Syntax: RST

Usage: This command resets the gosub stack to empty.

Restrictions: only allowed in programs

Remarks: This command will eliminate all gosubs that have been executed.

Example:

```
    DE1                     (define program 1)
    IN IV1                  (input variable value from key buffer)
    IF IV1>=0 GT5           (conditionally goto 5)
    PNT"-"                  (print string to display without carriage return and line feed)
    IV1=-1*IV1              (load integer variable 1)
 5  IV2=10                  (load integer variable 2 with pointer)
    GS10                    (unconditionally gosub 10)
    GT30                    (unconditionally goto 30)
10  IV5=IV1                 (load integer variable 5)
    IV1=IV1/10              (load integer variable 1)
    IV6=IV1*10              (load integer variable 6)
    IV5=IV5-IV6             (load integer variable 5)
    IVP2=IV5+48             (load pointed integer variable 2)
    IV2=IV2+1               (load integer variable 2 with pointer)
    IF IV2>16 GT20          (conditionally goto 20)
    IF IV1<>0 GS10          (conditionally gosub 10)
    IV2=IV2-1               (load integer variable 2 with pointer)
    ASCIVP2                 (send ASCII code to display)
    RET                     (return from gosub)
20  RST                     (reset gosub stack to empty)
    PRT"ERROR:"             (print string to display with carriage return and line feed)
    PRT"Number more than
            6 digits"       (print string to display with carriage return and line feed)
30  ED                      (end program 1 and exit program editor)
```

What will happen:

This program, once executed, will input an integer variable value from the key buffer. If the value is negative, the program will print a negative sign to the display, set the integer value positive, and continue to label 5, which sets the variable pointer to 10. It then goes to the subroutine at label 10, which stores the ASCII code of the ones digit in IV10, the ASCII code of the tens digit in IV11, and so on. If the number of digits is greater than 6, the program will goto label 20, which resets the gosub stack and prints out an error message. Otherwise, the program will then print the ASCII characters out to the screen to form the integer number IV1, return and goto label 30, which ends the program.

See Also: POP GS

RSV — REPORTS STRING VARIABLE VALUE

Type: diagnostic

Syntax: RSV*p1*

Parameters:

p1	
type	integer **or** integer variable **or** pointed integer variable
minimum	1
maximum	16

Usage: This command reports the value of string variable *p1*.

Restrictions: not allowed in programs or motion blocks

See Also: SV

RTC — REPORTS REAL TIME CLOCK VALUE

Type: diagnostic

Syntax: RTC

Usage: This command reports the value of the real time clock of the controller.

Restrictions: not allowed in programs or motion blocks

See Also: SRTC

RTM — REPORTS TIMER VALUE

Type: diagnostic

Syntax: RTM*p1*

Parameters: *p1*

type	integer
minimum	1
maximum	8

Usage: This command reports the value of timer *p1*.

Restrictions: not allowed in programs or motion blocks

Remarks: When the RTM command is executed, the value of timer *p1* will be given in seconds.

See Also: STM

RUC — REPORTS CONTROL OUTPUT

Type: diagnostic

Syntax: RUC

Usage: This command reports the control output.

Restrictions: not allowed in programs or motion blocks

Remarks: When the RUC command is executed, the value of the control output will be given in percent of the maximum.

See Also: AO

RVM — RETRIEVES USER MEMORY

Type: system

Syntax: RVM

Usage: This command is used to retrieve user memory from non-volatile memory.

Restrictions: not allowed in programs or motion blocks

Remarks: This command will execute only when the axis is stopped and no programs or motion blocks are executing.

See Also: SVM

RXP — REPORTS AUXILIARY POSITION

Type: diagnostic

Syntax: RXP

Usage: This command reports the auxiliary position. This is the position reading taken from the auxiliary position input of the axis.

Restrictions: not allowed in programs or motion blocks; extended command set only

Remarks: When the RXP command is executed, the auxiliary position will be given in auxiliary units.

See Also: RAP

RXT — REPORTS AUXILIARY POSITION CAPTURE

Type: diagnostic

Syntax: RXT

Usage: This command reports the auxiliary position capture. This is the position captured when the position capture is used to capture the auxiliary encoder input of the axis.

Restrictions: not allowed in programs or motion blocks; extended command set only

Remarks: If a position has not been captured, then the auxiliary position capture will be 0. After a position has been captured, the position can be reported using the RXT command, and then the auxiliary position capture will be set to 0 until a position is captured again. The auxiliary position capture will be given in auxiliary units.

See Also: RAT

RXV — REPORTS AUXILIARY VELOCITY

Type: diagnostic

Syntax: RXV

Usage: This command reports the auxiliary velocity. This is the velocity reading taken from the auxiliary position input of the axis.

Restrictions: not allowed in programs or motion blocks; extended command set only

Remarks: When the RXV command is executed, the auxiliary velocity will be given in auxiliary units/sec.

See Also: RAV

SAP — SETS AXIS POSITION REGISTER

Type: register

Syntax: SAP:*p1*

Parameters:

	p1
type	vexpr
units	units
minimum	-2,000,000,000 pulses
maximum	2,000,000,000 pulses

Usage: This command sets the axis position register.

Restrictions: not allowed in motion blocks

Remarks: The numerical values for the minimum and maximum of the parameter *p1* shown above are assuming that the axis unit ratio, AUR, is set at its default value of 1. If the axis unit ratio is set to a value other than 1, the maximum and minimum values will change appropriately (see AUR).

Example:

SAP:0	(set axis position register)
VL=10	(load velocity)
AC=40	(load acceleration rate)
AM=8	(load absolute move position)
RAN	(run to absolute move position)

What will happen:

By setting the axis position register, velocity, acceleration, absolute move position and issuing the RAN command, the axis will move 8 units in the forward direction.

See Also: AUR

SC — LOADS POWER SAVE CURRENT PERCENTAGE

Type: axis

Syntax: SC=*p1*

Parameters:

p1	
type	vexpr
units	%
default	60
minimum	0
maximum	100

Usage: This command loads the power save current percentage.

Restrictions: stepper only

Remarks: While the axis is in position, the continuous current value, CC, is reduced to the percentage loaded into SC. For example, if CC=50 and SC=20, the value of CC will be reduced to 10 percent while the axis is in position.

See Also: CC

SF — SECURES USER MEMORY SPACE

Type: system

Syntax: SF

Usage: This command secures user memory space so that programs and motion blocks cannot be modified or viewed. They can only be cleared, which is done by using the CLM command.

Restrictions: not allowed in programs or motion blocks

See Also: CLM

SI — APPLIES STEP INPUT

Type: motion

Syntax: SI:*p1*

Parameters:

	p1
type	vexpr
units	pulses
minimum	-16,000
maximum	16,000

Usage: This command applies a step input of *p1* pulses to the axis.

Restrictions: servo only; not allowed in motion blocks

Remarks: The step input cannot be larger than the following error bound, FB.

See Also: FB

SOP — SETS OFFSET POSITION REGISTER

Type: register

Syntax: SOP:*p1*

Parameters:

	p1
type	vexpr
units	units
minimum	-2,000,000,000 pulses
maximum	2,000,000,000 pulses

Usage: This command sets the offset position register.

Remarks: The numerical values for the minimum and maximum of the parameter *p1* shown above are assuming that the axis unit ratio, AUR, is set at its default value of 1. If the axis unit ratio is set to a value other than 1, the maximum and minimum values will change appropriately (see AUR).

Example:

SOP:0	(set offset position register)
VL=10	(load velocity)
AC=40	(load acceleration rate)
OM=10	(load offset move position)
RON	(run to offset move position)

What will happen:

By setting the offset position register, velocity, acceleration, offset move position and issuing the RON command, the axis will move 10 units in the forward direction.

See Also: AUR

SPA*n*B — LOADS SET POINT A BEGINNING

Type: input/output

Syntax: SPA*p1*B=*p2*

Parameters:

	p1	*p2*
type	integer	vexpr
units		units
default		OFF
minimum	1	-2,000,000,000 pulses
maximum	8	2,000,000,000 pulses

Usage: This command loads the set point A beginning position. In other words, it loads the position that set point A will turn on.

Remarks:

1. Set point A is designated as input/output 11 on the controller. Since it is a digital output, it is enabled by the DOE command.
2. Up to 8 pairs of set points can be defined, with *p1* being the pair designation.
3. Set point can be disabled by using the SPA*n*F command. This will set both SPA*n*B and SPA*n*E to the value "OFF".
4. The numerical values for the minimum and maximum of the parameter *p1* shown above are assuming that the axis unit ratio, AUR, is set at its default value of 1. If the axis unit ratio is set to a value other than 1, the maximum and minimum values will change appropriately (see AUR).

Example:

```
SAP:0          (set axis position register)
VL=5           (load velocity)
AC=50          (load acceleration)
AM=20          (load absolute move position)
SPA1B=5        (load set point A beginning 1)
SPA1E=10       (load set point A ending 1)
SPA2B=12       (load set point A beginning 2)
SPA2E=20       (load set point A ending 2)
RAN            (run to absolute move position)
```

What will happen:

First, the axis position register is set and the velocity, acceleration and absolute move position are loaded. Next, the set point A beginning 1 at 5 units and set point A ending 1 at 10 units are loaded. Then, set point A beginning 2 at 12 units and set point A ending 2 at 20 units are loaded. By issuing the RAN command, the axis will move 20 units in the forward direction. Set point A will turn on at 5 units, turn off at 10 units, turn on again at 12 units, and finally turn off at 20 units.

See Also: SPB*n*B, SPA*n*E, SPA*n*F, DOE, AUR

SPA*n*E — LOADS SET POINT A ENDING

Type: input/output

Syntax: SPA*p1*E=*p2*

Parameters:

	p1	*p2*
type	integer	vexpr
units		units
default		OFF
minimum	1	-2,000,000,000 pulses
maximum	8	2,000,000,000 pulses

Usage: This command loads the set point A ending position. In other words, it loads the position that set point A will turn off.

Remarks:

1. Set point A is designated as input/output 11 on the controller. Since it is a digital output, it is enabled by the DOE command.
2. Up to 8 pairs of set points can be defined, with *p1* being the pair designation.
3. Set point can be disabled by using the SPA*n*F command. This will set both SPA*n*B and SPA*n*E to the value "OFF".
4. The numerical values for the minimum and maximum of the parameter *p1* shown above are assuming that the axis unit ratio, AUR, is set at its default value of 1. If the axis unit ratio is set to a value other than 1, the maximum and minimum values will change appropriately (see AUR).

Example:

```
SAP:0       (set axis position register)
VL=5        (load velocity)
AC=50       (load acceleration)
AM=20       (load absolute move position)
SPA1B=5     (load set point A beginning 1)
SPA1E=10    (load set point A ending 1)
SPA2B=12    (load set point A beginning 2)
SPA2E=20    (load set point A ending 2)
RAN         (run to absolute move position)
```

What will happen:

First, the axis position register is set and the velocity, acceleration and absolute move position are loaded. Next, the set point A beginning 1 at 5 units and set point A ending 1 at 10 units are loaded. Then, set point A beginning 2 at 12 units and set point A ending 2 at 20 units are loaded. By issuing the RAN command, the axis will move 20 units in the forward direction. Set point A will turn on at 5 units, turn off at 10 units, turn on again at 12 units, and finally turn off at 20 units.

See Also: SPB*n*E, SPA*n*B, SPA*n*F, DOE, AUR

SPA*n*F — FORGETS SET POINT A

Type: input/output

Syntax: SPA*p1*F

Parameters:

	p1
type	integer
minimum	1
maximum	8

Usage: This command forgets set point A, i.e. turns set point A pair *p1* off.

Remarks: By using this command, SPA*n*B and SPA*n*E will be set to the value "OFF".

See Also: SPA*n*B
SPA*n*E

SPB*n*B — LOADS SET POINT B BEGINNING

Type: input/output

Syntax: SPB*p1*B=*p2*

Parameters:

	p1	*p2*
type	integer	vexpr
units		units
default		OFF
minimum	1	-2,000,000,000 pulses
maximum	8	2,000,000,000 pulses

Usage: This command loads the set point B beginning position. In other words, it loads the position that set point B will turn on.

Remarks:

1. Set point B is designated as input/output 12 on the controller. Since it is a digital output, it is enabled by the DOE command.
2. Up to 8 pairs of set points can be defined, with *p1* being the pair designation.
3. Set point B can be disabled by using the SPB*n*F command. This will set both SPB*n*B and SPB*n*E to the value "OFF".
4. The numerical values for the minimum and maximum of the parameter *p1* shown above are assuming that the axis unit ratio, AUR, is set at its default value of 1. If the axis unit ratio is set to a value other than 1, the maximum and minimum values will change appropriately (see AUR).

Example:

SAP:0	(set axis position register)
VL=5	(load velocity)
AC=50	(load acceleration)
AM=20	(load absolute move position)
SPB1B=5	(load set point B beginning 1)
SPB1E=10	(load set point B ending 1)
SPB2B=12	(load set point B beginning 2)
SPB2E=20	(load set point B ending 2)
RAN	(run to absolute move position)

What will happen:

First, the axis position register is set and the velocity, acceleration and absolute move position are loaded. Next, the set point B beginning 1 at 5 units and set point B ending 1 at 10 units are loaded. Then, set point B beginning 2 at 12 units and set point B ending 2 at 20 units are loaded. By issuing the RAN command, the axis will move 20 units in the forward direction. Set point B will turn on at 5 units, turn off at 10 units, turn on again at 12 units, and finally turn off at 20 units.

See Also: SPA*n*B, SPB*n*E, SPB*n*F, DOE, AUR

SPB*n*E — LOADS SET POINT B ENDING

Type: input/output

Syntax: SPB*p1*E=*p2*

Parameters:

	p1	*p2*
type	integer	vexpr
units		units
default		OFF
minimum	1	-2,000,000,000 pulses
maximum	8	2,000,000,000 pulses

Usage: This command load the sets point B ending position. In other words, it loads the position that set point B will turn off.

Remarks:

1. Set point B is designated as input/output 12 on the controller. Since it is a digital output, it is enabled by the DOE command.
2. Up to 8 pairs of set points can be defined, with *p1* being the pair designation.
3. Set point B can be disabled by using the SPB*n*F command. This will set both SPB*n*B and SPB*n*E to the value "OFF".
4. The numerical values for the minimum and maximum of the parameter *p1* shown above are assuming that the axis unit ratio, AUR, is set at its default value of 1. If the axis unit ratio is set to a value other than 1, the maximum and minimum values will change appropriately (see AUR).

Example:

SAP:0	(set axis position register)
VL=5	(load velocity)
AC=50	(load acceleration)
AM=20	(load absolute move position)
SPB1B=5	(load set point B beginning 1)
SPB1E=10	(load set point B ending 1)
SPB2B=12	(load set point B beginning 2)
SPB2E=20	(load set point B ending 2)
RAN	(run to absolute move position)

What will happen:

First, the axis position register is set and the velocity, acceleration and absolute move position are loaded. Next, the set point B beginning 1 at 5 units and set point B ending 1 at 10 units are loaded. Then, set point B beginning 2 at 12 units and set point B ending 2 at 20 units are loaded. By issuing the RAN command, the axis will move 20 units in the forward direction. Set point B will turn on at 5 units, turn off at 10 units, turn on again at 12 units, and finally turn off at 20 units.

See Also:

SPA*n*E	DOE
SPB*n*B	AUR
SPB*n*F	

SPB*n*F FORGETS SET POINT B

Type: input/output

Syntax: SPB*p1*F

Parameters: *p1*

type	integer
minimum	1
maximum	8

Usage: This command forgets set point B, i.e. turns set point B pair *p1* off.

Remarks: By using this command, SPB*n*B and SPB*n*E will be set to the value "OFF".

See Also: SPB*n*B
SPB*n*E

SPH — SETS PHASE POSITION TO ZERO

Type: register

Syntax: SPH

Usage: This command sets the phase position of the phase locked loop to zero. The phase position is the reference position of the phase locked loop.

Restrictions: extended command set only

See Also: RPP
PHL
PHO
PHE

SRTC — SETS REAL TIME CLOCK

Type: system

Syntax: SRTC:*p1*, *p2*, *p3*, *p4*, *p5*, *p6*

Parameters:

	p1	*p2*	*p3*	*p4*	*p5*	*p6*
units	year	month	day	hour	minute	second
minimum	1994	1	1	0	0	0
maximum	2060	12	31	23	59	59

Usage: This command sets the real time clock of the controller.

Restrictions: not allowed in programs or motion blocks

Example:

SRTC:1994,11,1,17,30,0 (sets real time clock to November 1, 1994, 5:30 PM)

See Also: RTC

ST — STOPS ALL MOTION

Type: motion

Syntax: ST

Usage: This command stops all motion.

Remarks: This command, once executed, will immediately decelerate the axis at the deceleration loaded.

Example:

```
VL=10      (load velocity)
DC=40      (load deceleration)
RFN        (run forward)
ST         (stops all motion)
```

What will happen:

By loading the velocity and deceleration and by issuing the RFN command, the axis will run in the forward direction. Then, by issuing the ST command, the axis will decelerate at 40 units/sec^2 to stop all motion.

See Also: HT
DC

STB — SET BOOLEAN VARIABLE

Type: variable

Syntax: STB*p1*

Parameters: *p1*

type	integer **or** integer variable **or** pointed integer variable
minimum	1
maximum	256

Usage: This command sets boolean variable *p1*, i.e. sets it to 1.

See Also: RSB
BV
RBV

STB...GT — SETS BOOLEAN VARIABLE AND IF VARIABLE WASN'T SET "GOTOS" LABEL

Type: program

Syntax: STB*p1* GT*p2*

Parameters:

	p1	*p2*
type	integer **or** integer variable **or** pointed integer variable	integer **or** integer variable **or** pointed integer variable
minimum	1	1
maximum	256	999

Usage: This command sets boolean variable *p1* and then checks to see if it was previously set. If boolean variable *p1* was not set, this command will cause the program to go to label *p2*.

Restrictions: not allowed out of programs

Example:

```
   DE1                    DE2                    (define program)
10 IV1=IV1+1           10 IV2=IV2+1              (load integer variable)
   IF IV1<1000 GT10       IF IV2<996 GT10        (conditionally goto 10)
   STB1 GT20              STB1 GT20              (set boolean variable 1 and if boolean variable 1
                                                  wasn't set, goto 20)
   GT10                   GT10                   (unconditionally goto 10)
20 PNT"IV1="           20 PNT"IV2="              (print string to display without carriage return and
                                                  line feed)
   OUT IV1                OUT IV2                (output variable value to the display)
   PRT""                  PRT""                  (print string to display with carriage return and line
                                                  feed)
   IV1=0                  IV2=0                  (load integer variable)
   RSB1                   RSB1                   (reset boolean variable 1)
   GT10                   GT10                   (unconditionally goto 10)
   ED                     ED                     (end program and exit program editor)
```

What will happen:

These two programs, when executed, will increment integer variables 1 and 2 until they reach 1000 and 996, respectively. The first program to finish this task will set boolean variable equal to 1, and since it was not previously set, it will goto the statement at label 20, which outputs the value to the display, loads 0 into the integer variable, and resets boolean variable 1. If one program finishes this task while the other is outputting the value to the display, the program will go back to label 10, increment the integer variable, and check again for boolean variable 1 to be reset.

See Also: IF...GT

STD — SETS DIGITAL OUTPUT

Type: input/output

Syntax: STD*p1*

Parameters:

p1	
type	integer
minimum	7
maximum	12

Usage: This command sets digital output *p1*, i.e. sets it to 1.

See Also: DOE
STDA
RSD
DO
RDO

STDA — SETS ALL DIGITAL OUTPUTS

Type: input/output

Syntax: STDA

Usage: This command sets all digital outputs, i.e. sets them all to 1.

See Also:
DOE
RSDA
STD
DO
RDO

STF — SETS FAULT

Type: system

Syntax: STF

Usage: This command faults the controller.

Restrictions: not allowed in motion blocks

See Also: RSF
RFC

STM — LOADS START TIME OF TIMER AND STARTS TIMER

Type: program

Syntax: STM*p1=p2*

Parameters:

	p1	*p2*
type	integer	vexpr
units		seconds
default		2,000,000.000
minimum	1	.001
maximum	8	2,000,000.000

Usage: This command loads the start time *p2* of timer *p1*, and loads timer *p1* with the start time *p2*, from which it will continuously count down to 0.

Example:

```
DE1     (define program 1)
VL=5    (load velocity)
AC=40   (load acceleration)
IM=10   (load incremental move position)
STM1=2  (load timer 1 with start time and start timer 1)
WT TM1  (wait for expression to be true)
RIN     (run to incremental move position)
ED      (end program 1 and exit program editor)
```

What will happen:

This program, when executed, will load the velocity, acceleration and incremental move position. It will then load timer 1 to start at 2 seconds and start timer 1. Lastly, it waits for the timer to count down from 2 seconds to 0. Then the RIN command will be executed, and the axis will move 10 units in the forward direction.

See Also: RTM

SUP — SUSPENDS MOTION

Type: program

Syntax: SUP

Usage: This command suspends all motion.

Restrictions: not allowed in motion blocks

Remarks: All motion will continue to be suspended until the RSM command is executed, which resumes the motion. However, if a motion command is issued while motion is suspended, the suspended motion will be eliminated.

See Also: RSM

SV — LOADS STRING VARIABLE

Type: variable

Syntax: SV*p1=p2*

Parameters:

	p1	*p2*
type	integer **or** integer variable **or** pointed integer variable	sexpr
minimum	1	0 characters
maximum	16	127 characters

Usage: This command loads string variable *p1* with the string expression *p2*.

Examples:

SV1="Test"	(load string variable 1 with "Test")
IV1=2	(load integer variable 1 with 2)
SVIV1=RGT11 RTC	(load string variable 2 with the "time" part of the real time clock value)
SV3=ASC32+ASC13	(load string variable 3 with a space followed by a carriage return)
SV4="Motion complete" +SV3	(load string variable 4 with "Motion complete" followed by a space and a carriage return)

See Also: RSV

SVM — SAVES USER MEMORY

Type: system

Syntax: SVM

Usage: This command is used to save user memory to non-volatile memory.

Restrictions: not allowed in programs or motion blocks

Remarks: This command will execute only when the axis is stopped and no programs or motion blocks are executing.

See Also: RVM

SXP — SETS AUXILIARY POSITION REGISTER

Type: register

Syntax: SXP:*p1*

Parameters:

	p1
type	vexpr
units	auxiliary units
minimum	-2,000,000,000 pulses
maximum	2,000,000,000 pulses

Usage: This command sets the auxiliary position register.

Restrictions: not allowed in motion blocks; extended command set only

Remarks: The numerical values for the default, minimum and maximum of this register are assuming that the auxiliary unit ratio, XUR, is set at its default value of 1. If the auxiliary unit ratio is set to a value other than 1, the default, maximum and minimum values will change appropriately (see XUR).

See Also: XUR

TC — LOADS TORQUE LIMIT CURRENT IN PERCENT OF CONTINUOUS CURRENT

Type: axis

Syntax: TC=*p1*

Parameters:

p1	
type	vexpr
units	%
default	100.0
minimum	1.0
maximum	100.0

Usage: This command loads the torque limit current in percent of the continuous current avaiable for the motor, CC.

Restrictions: servo only

Remarks: The torque limit is enabled by the TLE command.

See Also: TLE
CC
PC
SC

TGD — TOGGLES DIGITAL OUTPUT

Type: input/output

Syntax: TGD*p1*

Parameters:

	p1
type	integer
minimum	7
maximum	12

Usage: This command toggles digital output *p1*. In other words, it changes the present value of digital output *p1*.

See Also: DOE
RSD
STD
DO
RDO

TLE — ENABLES TORQUE LIMIT

Type: axis

Syntax: TLE=*p1*

Parameters:

p1	
type	boolean **or** bexpr
default	0
allowed values	0, 1

Usage: This command is used to enable the torque limit. If TLE is set to 1, then torque limit is enabled, and if TLE is set to 0, it is disabled.

Restrictions: servo only

See Also: TC

ULK — UNLOCKS INTERPRETER FROM PROGRAM

Type: program

Syntax: ULK

Usage: This command unlocks the interpreter from the program, which lets other currently suspended programs to execute concurrently.

Restrictions: only allowed in programs

Example:

```
    DE1          (define program 1)
1   WT TM1       (wait for expression to be true)
    LK           (lock interpreter to program)
    IF KY GT2    (conditionally goto 2)
    ULK          (unlock interpreter from program)
    GT1          (unconditionally goto 1)
2   ED           (end program and exit program editor)

    STM1=0.01    (load start time of timer 1 and start timer 1)
```

What will happen:

This program, once executed, will first wait for 10 ms. Then, it locks the interpreter and checks for KY to be true, i.e. for a character to be entered into the key buffer. If KY is true, then the program goes to the statement at label 2, which ends the program. If it is not, then it unlocks the interpreter and goes to the statement at label 1, which waits for 10 ms, etc.

See Also: LK

VL — LOADS VELOCITY

Type: register

Syntax: VL=*p1*

Parameters:

	p1
type	vexpr
units	units/sec
default	1 pulse/sec
minimum	1 pulse/sec
maximum	16,000,000 pulses/sec

Usage: This command loads the velocity rate. VL is used when the motion parameter, MP, is assigned to velocity.

Remarks: The numerical values for the default, minimum and maximum of the parameter *p1* shown above are assuming that the axis unit ratio, AUR, is set at its default value of 1. If the axis unit ratio is set to a value other than 1, the default, maximum and minimum values will change appropriately (see AUR).

Example:

```
SAP:0      (set axis position register)
VL=10      (load velocity)
AC=40      (load acceleration)
AM=12      (load absolute move position)
RAN        (run to absolute move position)
```

What will happen:

By setting the axis position register, velocity, acceleration, absolute move position and issuing the RAN command, the axis will move 12 units in the forward direction. It will accelerate at 40 units/sec^2 to a velocity of 10 units/sec, and then decelerate at 40 units/sec^2 to zero velocity.

See Also: MP
AUR

WT — WAITS FOR EXPRESSION TO BE TRUE

Type: program

Syntax: WT *p1*

Parameters:

	p1
type	bexpr
allowed values	0, 1

Usage: This command causes the program or motion block to wait for boolean expression *p1* to be true, i.e. evaluate to 1. Once it is, the next program or motion block statement will be executed.

Restrictions: not allowed out of programs or motion blocks

Example:

```
DE1                     (define program 1)
SAP:0                   (set axis position register)
VL=10                   (load velocity)
AC=40                   (load acceleration)
AM=0                    (load absolute move position)
IM=10                   (load incremental move position)
RIN                     (run to incremental move position)
WT IP                   (wait for expression to be true)
STM1=1                  (load start time of timer 1 and start timer 1)
WT TM1                  (wait for expression to be true)
RAN                     (run to absolute move position)
WT IP                   (wait for expression to be true)
PRT"Motion completed"   (print string to display with carriage return and line feed)
ED                      (end program 1 and exit program editor)
```

What will happen:

This program, once executed, will set the axis position register and load the velocity, acceleration, absolute move position, and incremental move position. Then, it issues the RIN command, which runs the axis 10 units in the forward direction. It then waits until the axis is in position. Next, it loads timer 1 with a start time of 1 second. The timer will then count down from 1 second to 0. Once it reaches 0, the RAN command is issued, which runs the axis 10 units in the reverse direction. It then waits until the axis is in position, and then it prints out "Motion completed" to the display.

See Also: WT...ON...GT

WT...ON...GT — WAITS FOR EXPRESSION TO BE TRUE OR ON CONDITION BECOMING TRUE "GOTOS" LABEL

Type: program

Syntax: WT *p1* ON *p2* GT*p3*

Parameters:

	p1	*p2*	*p3*
type	bexpr	boolean operand **or** boolean parameter	integer **or** integer variable **or** pointed integer variable
minimum			1
maximum			999
allowed values	0, 1	BV*n*, DO*n*, DI*n*, EG*n*, TM*n*, PX*n*, SRS*d*, SRA*d*, MB, KY, IP, TL, IO*d*, FI*d*, HSE, CIE, DSE, PFE, PWE, TLE, POE, OFE, DOE, AOE, DGE	

Usage: This statement causes the program to either wait for *p1* to become true, i.e. evaluate to 1, or to conditionally go to label *p3* if *p2* is true, i.e. evaluates to 1.

Restrictions: allowed only in programs

Remarks: Note that *p2* may also include an N (not), e.g. N MB. This specific example is a boolean that evaluates to 1 when no motion block is executing.

Example:

```
    DE1                     (define program 1)
    VL=5                    (load velocity)
    AC=40                   (load acceleration)
    IM=25                   (load incremental move position)
    RIN                     (run to incremental move position)
    WT IP ON KY GT5         (wait for expression to be true or on condition becoming true goto 5)
    PRT"Motion complete"    (print string to display with carriage return and line feed)
    GT10                    (unconditionally goto 10)
5   ST                      (stop all motion)
    WT IP                   (wait for expression to be true)
    PRT"Motion interrupted" (print string to display with carriage return and line feed)
10  ED
```

What will happen:

This program, once executed, loads the velocity, acceleration and incremental move position. It then issues the RIN command, which runs the axis 25 units in the forward direction. If a character goes into the key buffer before the axis is in position, the program will goto the statement at label 5, which stops all motion. It then prints "Motion interrupted" to the display and ends. If a character does not go into the key buffer before the axis is in position, the program will continue to the next statement, which prints "Motion complete". It then goes to the statement at label 10, which ends the program.

See Also: WT

X STEPS THROUGH PROGRAM/MOTION BLOCK

Type: program

Syntax: X *p1*

Parameters:

	p1
type	integer
minimum	1
maximum	999

Usage: This command steps *p1* lines through a program or motion block while in the program or motion block editor. Or, it steps through the execution of a program if not in the editor and single step mode is enabled, i.e. DGE is set to 1 and DGS is set to the program you wish to step through (see DGS).

Remarks: Note that *p1* is optional. If *p1* is not specified, a value of 1 will be assumed.

Example:

```
DE1                 (define program 1)
* VL=5
X                   (step through program)
* AC=40
X                   (step through program)
* IM=25
!                   (exit program editor)
*
```

See Also: DGE
DGS
DE
MB
L
LB
!

XFE — ENABLES AUXILIARY POSITION FEEDBACK

Type: axis

Syntax: XFE=*p1*

Parameters:

p1	
type	boolean
default	0
allowed values	0, 1

Usage: This command is used to enable auxiliary position feedback for servo motors. If XFE is set to 1, then the auxiliary position feedback is enabled, and if XFE is set to 0, it is disabled.

Restrictions: servo only; not allowed in programs or motion blocks; extended command set only

Remarks: When auxiliary position feedback is enabled, the axis position, AP, is set to the auxiliary encoder input. If the position feedback numerator, PFN, is zero, then only the auxiliary encoder input is used for position feedback. Otherwise, the primary position feedback is used for the primary position loop and the auxiliary encoder is used for the secondary position loop. When auxiliary position feedback is disabled, the axis position, AP, is set to the primary position feedback of the servo.

See Also: PFN
PFD
BL
CT
DB

XI — ASSIGNS AUXILIARY INPUT TYPE

Type: axis

Syntax: XI=*p1*

Parameters: *p1*
default: Q4
allowed values: PD, Q1, Q2, Q4

Usage: This command is used to assign the type of auxiliary position feedback. The possibilities are listed below:

PD - pulse/direction. This sets the feedback for a pulse input on channel A and a direction input on channel B.

Q1 - quadrature x1. This sets the feedback for two pulse waveforms in quadrature with a pulse multiplier of 1.

Q2 - quadrature x2. This sets the feedback for two pulse waveforms in quadrature with a pulse multiplier of 2.

Q4 - quadrature x4. This sets the feedback for two pulse waveforms in quadrature with a pulse multiplier of 4.

Restrictions: not allowed in programs or motion blocks; extended command set only

See Also: FR
XFE

XPL — LOADS AUXILIARY POSITION LENGTH

Type: floating point

Syntax: XPL=*p1*

Parameters:

p1	
type	integer
units	auxiliary units
default	2,000,000,000 pulses
minimum	500 pulses
maximum	2,000,000,000 pulses

Restrictions: not allowed in programs or motion blocks; extended command set only

Usage: This command loads the auxiliary position length. This is actually half the auxiliary position register length. The auxiliary position register counts from -XPL units to XPL-1 units.

Remarks: The numerical values for the default, minimum and maximum of this register are assuming that the auxiliary unit ratio, XUR, is set at its default value of 1. If the auxiliary unit ratio is set to a value other than 1, the default, maximum and minimum values will change appropriately (see XUR).

See Also: XUR

XUR — LOADS AUXILIARY UNIT RATIO

Type: axis

Syntax XUR=*p1*

Parameters:

p1	
type	integer
units	pulses/auxiliary unit
default	1
minimum	1
maximum	1,000,000

Usage: This command loads the auxiliary unit ratio.

Restrictions: extended command set only; not allowed in programs or motion blocks

Remarks: The numerical values for the default, minimum and maximum of all parameters whose units are auxiliary units are assuming that the auxiliary unit ratio, XUR, is set at its default value of 1. If the auxiliary unit ratio is set to a value other than 1, the maximum and minimum values will be divided by the auxiliary unit ratio. For example, if the maximum value of a parameter is 2,000,000,000 pulses and the auxiliary unit ratio is set to 4096, the new maximum of that parameter will be (2,000,000,000 pulses)/(4096 pulses/ auxilary unit) = 488,281 auxiliary units.

Appendix B

Expression Charts

Appendix B: Expression Charts

Chart 1: Expression Types

Type	bexpr (boolean expression)	vexpr (variable expression)	sexpr (string expression)
Syntax	syntax 1: *p1 p2 p3 p4 p5* syntax 2: *p1 p2 p3*	*p1 p2 p3 p4*	*p1 p2 p3 p4*
Parameters	syntax 1: *p1, p4* = unary boolean operator *p2, p5* = boolean variable or boolean operand *p3* = binary boolean operator syntax 2: *p1* = integer, floating point or string variable *p2* = relational operator *p3* = integer, floating point or string variable or integer or floating point number	*p1* = unary boolean or unary variable operator *p2, p4* = boolean, integer, floating point or string variable or numerical operand or integer or floating point number *p3* = binary boolean or mathematical or shift operator	*p1* = unary string operator *p2, p4* = boolean, integer, floating point or string variable or string operand or string *p3* = concatenation operator
Remarks	1. In syntax 1, *p1* and *p4* are optional. 2. In syntax 1, *p3* and *p5* together are optional.	1. *p1* is optional. 2. *p3* and *p4* together are optional.	1. *p1* is optional. 2. *p3* and *p4* together are optional.
Examples	syntax 1: BV1 AND N BV2 DO7 N SRS0 OR MB syntax 2: IV1 > 5 FV1 <= FV2 SV1 = SVIV1	FV1 + IV1 AP - CP IV1 AND 63 SIN 45 ABS IV2 * IV3 AO / 10	"Drilling complete" ASC13 + ASC10 RGT11 RTC "Axis position=" + AP LFT SV1 IV1

Chart 2: Operands

Operand	Type	Description	Range/Allowed Values
BV*n*	boolean variable	boolean variable *n*.	*n*: 1 to 256 BV*n*: 0, 1
IV*n*	integer variable	Integer variable *n*.	*n*: 1 to 4096 - FP*2 IV*n*: -2,147,483,648 to 2,147,483,647
IVP*n*	integer variable	The integer variable pointed to by integer variable *n*.	*n*: 1 to 4096 - FP*2 IVP*n*: -2,147,483,648 to 2,147,483,647
FV*n*	floating point variable	Floating point variable *n*.	*n*: 1 to FP FV*n*: 1.5 x 10^{-39} (absolute value) to 1.7 x 10^{38} (absolute value)
FVP*n*	floating point variable	The floating point variable pointed to by integer variable *n*.	*n*: 1 to 4096 - FP*2 FVP*n*: 1.5 x 10^{-39} (absolute value) to 1.7 x 10^{38} (absolute value)
SV*n*	string variable	String variable *n*.	*n*: 1 to 16 SV*n*: 0 to 127 characters

Operand	Type	Description	Range/Allowed Values
SVIV*n*	string variable	The string variable pointed to by integer variable *n*.	*n*: 1 to 4096 - FP*2 SVIV*n*: 0 to 127 characters
SVIVP*n*	string variable	The string variable pointed to by the integer variable pointed to by integer variable *n*.	*n*: 1 to 4096 - FP*2 SVIVP*n*: 0 to 127 characters
ASC*n*	string operand	The character represented by ASCII code *n*.	*n*: 0 to 255 ASC*n*: ASCII character set
ASCIV*n*	string operand	The character represented by the ASCII code loaded in integer variable *n*.	*n*: 1 to 4096 - FP*2 ASCIV*n*: ASCII character set
ASCIVP*n*	string operand	The character represented by the ASCII code loaded in the integer variable pointed to by integer variable *n*.	*n*: 1 to 4096 - FP*2 ASCIVP*n*: ASCII character set
ASC SV*n*	numerical operand	ASCII value of the first character of string variable *n*.	*n*: 1 to 16 ASC SV*n*: 0 to 255
ASC SVIV*n*	numerical operand	ASCII value of the first character of the string variable pointed to by integer variable *n*.	*n*: 1 to 4096 - FP*2 ASC SVIV*n*: 0 to 255
ASC SVIVP*n*	numerical operand	ASCII value of the first character of the string variable pointed to by the integer variable pointed to by integer variable *n*.	*n*: 1 to 4096 - FP*2 ASC SVIVP*n*: 0 to 255
LEN SV*n*	numerical operand	The length of string variable *n*.	*n*: 1 to 16 LEN SV*n*: 0 to 127
LEN SVIV*n*	numerical operand	The length of the string variable pointed to by integer variable *n*.	*n*: 1 to 4096 - FP*2 LEN SVIV*n*: 0 to 127
LEN SVIVP*n*	numerical operand	The length of the string variable pointed to by the integer variable pointed to by integer variable *n*.	*n*: 1 to 4096 - FP*2 LEN SVIVP*n*: 0 to 127
DO	numerical or string operand	A number that represents the states of the digital outputs. If used in a variable expression, it is a decimal number, and if used in a string expression, it is a hexadecimal number.	64 to 4,032 (vexpr) 040 to FC0 (sexpr)
DO*n*	boolean operand	The state of digital output *n*.	*n*: 7 to 12 DO*n*: 0, 1
DI	numerical or string operand	A number that represents the states of the digital inputs. If used in a variable expression, it is a decimal number, and if used in a string expression, it is a hexadecimal number.	0 to 4,095 (vexpr) 000 to FFF (sexpr)
DI*n*	boolean operand	The state of digital input *n*.	*n*: 1 to 12 DI*n*: 0, 1

Operand	Type	Description	Range/Allowed Values
EG	numerical or string operand	A number that represents the states of the positive edge sensitive digital inputs. If used in a variable expression, it is a decimal number, and if used in a string expression, it is a hexadecimal number.	0 to 4,095 (vexpr) 000 to FFF (sexpr)
EG*n*	boolean operand	The state of positive edge sensitive digital input *n*.	*n*: 1 to 12 EG*n*: 0, 1
AO	numerical or string operand	The voltage value of the analog output.	-10.000 to 10.000 volts
AI[1]	numerical or string operand	The voltage value of the analog input.	-10.000 to 10.000 volts
RTC	numerical or string operand	The value of the real time clock. If used in a variable expression, it is the number of seconds from 1/1/94, and if used in a string expression, it is a formatted output showing the date and time.	0 to 2,147,483,647 (vexpr) January 1, 1994 12:00:00 AM to December 31, 2060 11:59:59 PM (sexpr)
TM*n*	boolean or numerical or string operand	The status of timer *n*. If used in a boolean expression, it is equal to 1 when the timer has timed out. If a used in a variable or string expression, it represents the time left in timer *n*.	*n*: 1 to 8 TM*n*: 0, 1 (bexpr) 0 to 2,000,000.000 seconds (vexpr or sexpr)
PS*n*	numerical or string operand	The status of program *n*. This is the value of the program status register. If used in a variable expression, it is a decimal number, and if used in a string expression, it is a hexadecimal number.	*n*: 1 to 4 PS*n*: 0 to 65,535 (vexpr) 0000 to FFFF (sexpr)
PX*n*	boolean operand	Program *n* executing. This is equal to 1 if program *n* is executing, and equal to 0 if program *n* is not executing.	*n*: 1 to 4 PX*n*: 0, 1
SE*n*	boolean operand	String to number conversion error for program *n*. This is equal to 1 if either bit 4 or bit 5 of the program status register is set.	*n*: 1 to 4 SE*n*: 0, 1
SRS	numerical or string operand	The value of the system status register. If used in a variable expression, it is a decimal number, and if used in a string expression, it is a hexadecimal number.	0 to 65,535 (vexpr) 0000 to FFFF (sexpr)
SRS*d*	boolean operand	The value of system status register bit *d*. *d* is a hexadecimal digit.	*d*: 0 to F SRS*d*: 0, 1
SRA	numerical or string operand	The value of the axis status register. If used in a variable expression, it is a decimal number, and if used in a string expression, it is a hexadecimal number.	0 to 65,535 (vexpr) 0000 to FFFF (sexpr)
SRA*d*	boolean operand	The value of axis status register bit *d*. *d* is a hexadecimal digit.	*d*: 0 to F SRA*d*: 0, 1

Operand	Type	Description	Range/Allowed Values
MB	boolean operand	Motion block executing. This is equal to 1 if a motion block is executing, and equal to 0 if no motion block is executing.	0, 1
KY	boolean operand	Character in key buffer. This is equal to 1 if the key buffer has one or more characters in it, and equal to 0 if the key buffer is empty.	0, 1
IP	boolean operand	Axis in position. This is equal to 1 if the axis is in position, and equal to 0 if the axis is not in position.	0, 1
TL	boolean operand	At torque limit. This is equal to 1 if the torque limit has been reached, and equal to 0 if it has not been reached.	0, 1
IO	numerical or string operand	The value of the I/O register. If used in a variable expression, it is a decimal number, and if used in a string expression, it is a hexadecimal number.	0 to 65,535 (vexpr) 0000 to FFFF (sexpr)
IO*d*	boolean operand	The value of I/O register bit *d*. *d* is a hexadecimal digit.	*d*: 0 to F IO*d*: 0, 1
FI	numerical or string operand	The value of the fault input register. If used in a variable expression, it is a decimal number, and if used in a string expression, it is a hexadecimal number.	0 to 65,535 (vexpr) 0000 to FFFF (sexpr)
FI*d*	boolean operand	The value of fault input register bit *d*. *d* is a hexadecimal digit.	*d*: 0 to F FI*d*: 0, 1
FC	numerical or string operand	The value of the fault code register. If used in a variable expression, it is a decimal number, and if used in a string expression, it is a hexadecimal number.	0 to 4,294,967,295 (vexpr) 00000000 to FFFFFFFF (sexpr)
FC*n*	boolean operand	The value of the fault code register bit *n*. *n* is an integer.	*n*: 0 to 31 FC*n*: 0, 1
CP	numerical or string operand	Command position. This is the position that the controller wants the axis to be at.	-2,000,000,000 to 2,000,000,000 pulses*
AP	numerical or string operand	Axis position. This is the actual position of the axis.	-2,000,000,000 to 2,000,000,000 pulses*
AT	numerical or string operand	Axis position capture. This is the position of the axis that was captured with the position capture input.	-2,000,000,000 to 2,000,000,000 pulses*
AV	numerical or string operand	Axis velocity. This is the velocity of the axis.	1 to 16,000,000 pulses/sec*
XP[1]	numerical or string operand	Auxiliary position. This is the position of the auxiliary input.	-2,000,000,000 to 2,000,000,000 pulses**

Operand	Type	Description	Range/Allowed Values
XT[1]	numerical or string operand	Auxiliary position capture. This is the position of the auxiliary input that was captured with the position capture input .	-2,000,000,000 to 2,000,000,000 pulses**
XV[1]	numerical or string operand	Auxiliary velocity. This is the velocity of the auxiliary input.	1 to 16,000,000 pulses/sec**
UC	numerical or string operand	Control output in percent of the maximum control output.	-100.00 to 100.00 percent
FE	numerical or string operand	Following error. This is the difference between the command position, CP, and the axis position, AP.	0 to 16,000 pulses*
PP[1]	numerical or string operand	Phase position. This is the reference position of the phase locked loop.	-PHL/2 to PHL/2 - 1 pulses
PE[1]	numerical or string operand	Phase error. This is the difference between the desired reference position and the reference position that was captured when the position capture input became active.	0 to 32,000 pulses
PM[1]	numerical or string operand	Phase multiplier. This is the ratio between the reference input and the axis output.	0 to 10,000
RR	numerical or string operand	Resolver reading.	0 to 4095
ATE	boolean operand	Auto retrieve enable.[2]	0, 1
FP	numerical or string operand	Floating point variable allocation.[2]	0 to 2048
BD	numerical or string operand	Baud rate of the serial port of the controller.[2]	1200, 9600, 19200, 38400 baud
HSE	boolean operand	Handshake protocol enable.[2]	0, 1
CIE	boolean operand	Computer interface format enable.[2]	0, 1
DSE	boolean operand	Display format enable.[2]	0, 1
XFE[1]	boolean operand	Auxiliary position feedback enable.[2]	0, 1
PFE	boolean operand	Position feedback enable.[2]	0, 1
PWE	boolean operand	Position register wrap enable.[2]	0, 1
AUR	numerical or string operand	Axis unit ratio.[2]	1 to 1,000,000 pulses/unit
XUR[1]	numerical or string operand	Auxiliary unit ratio.[2]	1 to 1,000,000 pulses/unit
APL	numerical or string operand	Axis position length. This is actually half the axis position register length.[2]	500 to 2,000,000,000 pulses*
XPL[1]	numerical or string operand	Auxiliary position length. This is actually half the auxiliary position register length.[2]	500 to 2,000,000,000 pulses**

Operand	Type	Description	Range/Allowed Values
FO	numerical or string operand	Forward software overtravel limit.[2]	-2,000,000,000 to 2,000,000,000 pulses*
RO	numerical or string operand	Reverse software overtravel limit.[2]	-2,000,000,000 to 2,000,000,000 pulses*
FB	numerical or string operand	Following error bound.[2]	0 to 16,000 pulses*
PB	numerical or string operand	In-position band.[2]	0 to 16,000 pulses*
FR	numerical or string operand	Axis feedback resolution.[2]	500 to 1,000,000 pulses/rev
AR	numerical or string operand	Amplitude of resolver excitation.[2]	1 to 4
PR	numerical or string operand	Motor poles to resolver poles ratio.[2]	1 to 16
CO	numerical or string operand	Commutation angle offset.[2]	-180.0 to 180.0
CA	numerical or string operand	Commutation angle advance.[2]	-90.0 to 90.0
CC	numerical or string operand	Continuous current in percent of maximum.[2]	1 to 100 percent
PC	numerical or string operand	Peak current in percent of maximum.[2]	1 to 100 percent
SC	numerical or string operand	Power save current in percent of maximum.[2]	0 to 100 percent
TC	numerical or string operand	Torque limit current in percent of maximum.[2]	1 to 100 percent
TLE	boolean operand	Torque limit enable.[2]	0, 1
POE	boolean operand	Power output stage enable.[2]	0, 1
KP	numerical or string operand	Proportional control gain.[2]	0 to 8,000
KI	numerical or string operand	Integral control gain.[2]	0 to 64,000
KD	numerical or string operand	Derivative control gain.[2]	0 to 8,000
KA	numerical or string operand	Acceleration feedforward constant.[2]	0 to 64,000
FT	numerical or string operand	Filter time constant.[2]	0 to 5
CT	numerical or string operand	Correction time constant.[2]	0.001 to 4.000 seconds
BL	numerical or string operand	Backlash constant.[2]	0 to 16,000 pulses
DB	numerical or string operand	Deadband constant.[2]	0 to 16,000 pulses*

Operand	Type	Description	Range/Allowed Values
STM*n*	numerical or string operand	Start time of timer *n*.[2]	.001 to 2,000,000.000 seconds
AM	numerical or string operand	Absolute move position.[2]	-2,000,000,000 to 2,000,000,000 pulses*
OM	numerical or string operand	Offset move position.[2]	-2,000,000,000 to 2,000,000,000 pulses*
IM	numerical or string operand	Incremental move position.[2]	-2,000,000,000 to 2,000,000,000 pulses*
PSP[1]	numerical or string operand	Pulse motion start position.[2]	-2,000,000,000 to 2,000,000,000 pulses**
JK	numerical or string operand	Jerk percentage.[2]	0 to 100 percent
AC	numerical or string operand	Acceleration.[2]	100 to 1,000,000,000 pulses/sec^2*
DC	numerical or string operand	Deceleration.[2]	100 to 1,000,000,000 pulses/sec^2*
VL	numerical or string operand	Velocity.[2]	1 to 16,000,000 pulses/sec*
ACP	numerical or string operand	Acceleration percentage.[2]	1 to 99 percent
DCP	numerical or string operand	Deceleration percentage.[2]	1 to 99 percent
MVP[1]	numerical or string operand	Move pulses.[2]	1 to 2,000,000,000 pulses**
MVT	numerical or string operand	Move time.[2]	.005 to 2,000,000.000 seconds
RN[1]	numerical or string operand	Ratio numerator.[2]	-10,000 to 10,000
RD[1]	numerical or string operand	Ratio denominator.[2]	1 to 10,000
PHL[1]	numerical or string operand	Phase length.[2]	500 to 64,000 pulses
PHO[1]	numerical or string operand	Phase offset.[2]	-32,000 to 32,000 pulses
PHN[1]	numerical or string operand	Phase numerator.[2]	-10,000 to 10,000
PHD[1]	numerical or string operand	Phase denominator.[2]	1 to 10,000
PHG[1]	numerical or string operand	Phase gain.[2]	0 to 255
PHZ[1]	numerical or string operand	Phase zero.[2]	0 to 255
PHB[1]	numerical or string operand	Phase error bound.[2]	0 to 32,000 pulses

Operand	Type	Description	Range/Allowed Values
PHT[1]	numerical or string operand	Phase lockout time.[2]	.001 to 4.000 seconds
RIE[1]	boolean operand	Ratio input enable.[2]	0, 1
PHE[1]	boolean operand	Phase locked loop enable.[2]	0, 1
DFT	numerical or string operand	Digital input filter time.[2]	.001 to 4.000 seconds
DIF	numerical or string operand	Which digital inputs have been assigned a digital input filter.[2] If used in a variable expression, it is a decimal number, and if used in a string expression, it is a hexadecimal number.	0 to 4,095 (vexpr) 000 to FFF (sexpr)
OFE	boolean operand	Fault on output fault enable.[2]	0, 1
DOE	boolean operand	Digital output enable.[2]	0, 1
AOE	boolean operand	Analog output enable.[2]	0, 1
SPA*n*B	numerical or string operand	Set point A beginning *n*.[2]	*n*: 1 to 8 SPA*n*B: -2,000,000,000 to 2,000,000,000 pulses*
SPB*n*B	numerical or string operand	Set point B beginning *n*.[2]	*n*: 1 to 8 SPB*n*B: -2,000,000,000 to 2,000,000,000 pulses*
SPA*n*E	numerical or string operand	Set point A ending *n*.[2]	*n*: 1 to 8 SPA*n*E: -2,000,000,000 to 2,000,000,000 pulses*
SPB*n*E	numerical or string operand	Set point B ending *n*.[2]	*n*: 1 to 8 SPB*n*E: -2,000,000,000 to 2,000,000,000 pulses*
FW	numerical or string operand	Field width of output variable on display.[2]	0 to 30
DP	numerical or string operand	Number of places after decimal point on display.[2]	0 to 10
DGE	boolean operand	Diagnostics enable.[2]	0, 1
DGS	numerical or string operand	Program in single step mode.[2]	0 to 4

[1] This is only available in the extended IMC-C.
[2] See command summary for further description.
* The possible values depend on the setting of the axis unit ratio, AUR. See command summary.
** The possible values depend on the setting of the auxiliary unit ratio, XUR. See command summary.

Chart 3: Operators

Operator	Type	Description
N	unary boolean operator	Performs a logical "not" on a boolean operand, and a bitwise logical "not" on a variable operand.
AND	binary boolean operator	Performs a logical "and" on a boolean operand, and a bitwise logical "and" on a variable operand.

Operator	Type	Description
OR	binary boolean operator	Performs a logical "or" on a boolean operand, and a bitwise logical "or" on a variable operand.
XOR	binary boolean operator	Performs a logical "exclusive or" on a boolean operand, and a bitwise logical "exclusive or" on a variable operand.
LFT	shift operator	Performs an arithmetic shift left on the first integer operand by the number of places specified in the second integer operand.
RGT	shift operator	Performs an arithmetic shift right on the first integer operand by the number of places specified in the second integer operand.
+	concatenation operator	Performs a concatenation of two string operands.
>	relational operator	Greater than.
>=	relational operator	Greater than or equal to.
=	relational operator	Equal to.
<>	relational operator	Not equal to.
<=	relational operator	Less than or equal to.
<	relational operator	Less than.
+	mathematical operator	Add.
-	mathematical operator	Subtract.
*	mathematical operator	Multiply.
/	mathematical operator	Divide.
ABS	unary variable operator	Performs an absolute value operation on a variable operand.
SQR	unary variable operator	Performs a square root operation on a variable operand.
SIN	unary variable operator	Performs a sine operation on a variable operand, where the operand is in degrees.
COS	unary variable operator	Performs a cosine operation on a variable operand, where the operand is in degrees.
TAN	unary variable operator	Performs a tangent operation on a variable operand, where the operand is in degrees.
ATN	unary variable operator	Performs a arctangent operation on a variable operand, where the result is in degrees.
ETX	unary variable operator	Performs a "e to the x" operation on a variable operand. It takes the number e to the power of the variable operand.
LGN	unary variable operator	Performs a natural log operation on a variable operand.
LFT*n*	unary string operator	Takes the *n* leftmost characters of the string operand.
LFTIV*n*	unary string operator	Takes the IV*n* leftmost characters of the string operand.
LFTIVP*n*	unary string operator	Takes the IVP*n* leftmost characters of the string operand.
RGT*n*	unary string operator	Takes the *n* rightmost characters of the string operand.
RGTIV*n*	unary string operator	Takes the IV*n* rightmost characters of the string operand.
RGTIVP*n*	unary string operator	Takes the IVP*n* rightmost characters of the string operand.
LWR	unary string operator	Makes all the characters in the string operand lowercase.
UPR	unary string operator	Makes all the characters in the string operand uppercase.

Appendix C

Message Charts

Appendix C: Message Charts

Chart 1: Command Messages

Number	Command Message	Possible Cause(s)	Possible Solution(s)
6	RECEIVE ERROR	A character that was entered was not received correctly by the controller.	► The controller expects 1 start bit, 1 stop bit, 7 data bits and odd parity. Check to make sure that these settings are correct on the terminal you are using. Also, check the baud rate setting of the terminal. ► Check the COM port connection from the terminal to the controller.
8	LABEL OUT OF RANGE	The program label entered as part of a program statement is less than 1 or greater than 999.	► Reenter the label making sure that it is a number 1 through 999.
9	INVALID COMMAND	The command entered was not recognized by the controller.	► The command was misspelled or not in capital letters. Reenter the command making sure that it is spelled correctly and in all capitals. ► The command was invalid. Reenter the command making sure that it is a valid command.
10	INVALID ASSIGNMENT	The assignment entered was not valid for the command entered.	► The assignment was misspelled or not in capital letters. Reenter the command making sure that the assignment is spelled correctly and in all capitals. ► The assignment was invalid. Reenter the command making sure that the assignment you are using is valid for the command you are entering.
11	INVALID OPERAND	The operand entered was not valid for the command entered.	► The operand was misspelled or not in capital letters. Reenter the command making sure that the operand is spelled correctly and in all capitals. ► The operand was invalid. Reenter the command making sure that the operand you are using is valid for the command you are entering.
12	INVALID DIGIT	The number entered as a parameter for the command contained an invalid digit.	► Reenter the command making sure that the parameter does not contain an invalid digit.
13	VALUE OUT OF RANGE	The value entered as a parameter was out of the range specified for the command entered.	► Reenter the command making sure that the parameter is within the range specified for the command. See the "Parameters" section of the command summary for the allowed range.
14	TOO MANY DECIMAL PLACES	The value entered as a parameter had more decimal places than allowed for the command entered.	► Reenter the command making sure that the parameter does not have too many decimal places.
15	STRING TOO LONG	The string entered was longer than 127 characters.	► Reenter the command/string making sure that the string is no longer than 127 characters.

Number	Command Message	Possible Cause(s)	Possible Solution(s)
16	DUPLICATE LABEL	The program label entered as part of a program statement was already in the current program.	► Reenter the program statement making sure that the label does not already exist in the program.
17	NONEXISTENT LABEL	The LB command was entered in the program editor with a nonexistent label.	► Reenter the LB command making sure that the label exists in the program.
18	OUT OF PROGRAM MEMORY	The controller has run out of memory available for programs and motion blocks.	► Delete any programs or motion blocks that are not currently being used.
19	NOT READY FOR COMMAND	The controller was not ready to accept the command entered because it was executing an operation that cannot be interrupted by that command.	► Wait until the operation finishes or stop it completely; programs can be killed with the KL or KLA command and any motion can be stopped with the ST or HT command. For specific information about the command you are using, see the "Remarks" section of the command summary.
20	COMMAND NOT ALLOWED	The command entered in the program/motion block editor is not allowed in a program/motion block or a the command entered in immediate mode is only allowed in a program and/or motion block.	► For specific information about the command you are using, see the "Restrictions" section of the command summary.
21	NO PROGRAM FAULT	The FLT command was entered without there being a program fault.	► If the controller is faulted, the RFC command can be used to show what fault has occurred.
22	INVALID VARIABLE POINTER	The pointer loaded in an integer variable was out of the range of variables available.	► Reenter the pointer making sure that it is in the range of the type of variable accessed.
23	MISMATCHED OPERANDS AND OPERATOR	The calculation specified by the boolean, variable or string expression cannot be done in the format of the variable or parameter to be loaded.	► Use N, AND, OR and XOR with boolean and integer operands only. ► Use LFT and RGT with integer operands only. ► Use SIN, COS, TAN, ATN, ETX, and LGN with floating point operands only.
24	MATHEMATICAL OVERFLOW	The result of the variable expression entered was outside of the allowed bounds of the variable.	► Reenter the variable expression making sure that the operation will never go outside the allowed bounds of the variable. ► Consider using a double integer variable expression if you are working with very large numbers.
25	MATHEMATICAL DATA ERROR	The result of the variable expression entered cannot be stored as a number.	► Make sure that the SQR and LGN operators never have negative operands. ► Make sure that a divide by zero operation will never occur.

Number	Command Message	Possible Cause(s)	Possible Solution(s)
26	INVALID MOTION	An attempt was made to execute a run command while a home command was executing or vice versa; the combination of motion parameters define a motion that cannot be executed; or a motion command or motion block was executed when the controller was faulted.	► Make sure that the homing operation stops before executing a run command, or make sure that the motor is stopped before executing a home command. ► Make sure that the motion parameters define a motion that can be executed. For specific information about the parameters you are using, see the command summary. ► Make sure the controller is not faulted when executing a motion command or motion block.
27	SWITCH MOTOR LEADS	The MS command was entered and the controller decided from its calculations that two motor leads should be switched.	► Switch two of the motor leads.
28	BAD POLES RATIO	The MS command was entered and the controller calculated the motor poles to resolver poles ratio to be less than 1 or greater than 16.	► Use a different motor.
30	TORQUE TO INERTIA RATIO TOO LOW	The KS command was entered and the controller calculated the torque to inertia ratio of the axis to be less than 125 radians/sec^2.	► Auto tuning with the KS command will not work. Use the expressions for KA, KD, KI, KP and FT to calculate values if possible. See command summary.
31	TORQUE TO INERTIA RATIO TOO HIGH	The KS command was entered and the controller calculated the torque to inertia ratio of the axis to be greater than 125,000 radians/sec^2.	► Auto tuning with the KS command will not work. Use the expressions for KA, KD, KI, KP and FT to calculate values if possible. See command summary.
32	TORQUE RESPONSE NON-LINEAR	The KS command was entered and the controller could not calculate the control constants because the motor did not respond linearly.	► Auto tuning with the KS command will not work. Use the expressions for KA, KD, KI, KP and FT to calculate values if possible. See command summary.
33	Enter password:	The PW command has been entered and the controller is waiting for the password to be entered.	► Enter the password.
34	Password accepted	The PW command has been entered and the correct password entered, or the PWC command has been entered and the new password entered correctly.	► Continue with normal operation.

Number	Command Message	Possible Cause(s)	Possible Solution(s)
35	Invalid password - access denied	The PW or PWC command has been entered and the password entered is incorrect.	► Enter the PW or PWC command again and enter the correct password.
36	Enter old password:	The PWC command has been entered and the controller is waiting for the old password to be entered.	► Enter the old password, i.e. the password that you have used up to this point.
37	Enter new password:	The PWC command has been entered and the controller is waiting for the new password to be entered.	► Enter the new password, i.e. the password that you wish to have.
38	Enter new password again to verify:	The PWC command has been entered and the controller is waiting for the new password to be entered again for verification purposes.	► Enter the new password again.
39	Invalid password - Password unchanged	The PWC command has been entered, and either the new password entered is invalid or the new password entered the second time does not match the one entered the first time.	► Enter the PWC command again to start over. Make sure that the new password is no longer than 10 characters.
40	Retrieving user memory...	The RVM command has been entered and the controller is in the process of retrieving user memory.	► Wait for user memory to be retrieved.
41	User memory retrieved	The RVM command has been entered and the controller has retrieved user memory.	► Continue with normal operation.
42	Saving user memory...	The SVM command has been entered and the controller is in the process of saving user memory.	► Wait for user memory to be saved.
43	User memory saved	The SVM command has been entered and the controller has saved user memory.	► Continue with normal operation.

Chart 2: Fault Code Messages

Bit	Fault Code Message	Possible Cause(s)	Possible Solution(s)
0	Power Failure	A power failure has occurred. This fault always occurs when the controller is powered up.	► Use the RSF command to reset the fault condition.
2	Software Fault	The STF command was executed.	► Use the RSF command to reset the fault condition.
3	Lost Enable	The enable input was deactivated.	► Reactivate the enable input and use the RSF command the reset the fault condition.
4	Digital Output Fault	The digital input associated with the digital output did not detect a change of state in the output after executing the RSD(A) or STD(A) command, and the output fault enable, OFE, is enabled.	► Check that the output common is connected to power return and the input common is connected to power supply or vice versa, depending on whether you have a sinking or sourcing configuration. ► Check that the output is not shorted.
5	Invalid Command	An attempt was made by the program to execute the EXSV command, but the command stored in the string variable was not recognized by the controller.	► The command is misspelled or not in capital letters. Reenter the command in the program editor making sure that it is spelled correctly and in all capitals. ► The command is invalid. Reenter the command in the program editor making sure that it is a valid command.
6	Command Not Allowed	An attempt was made by the program to execute the EXSV command, but the command stored in the string variable is not allowed in a program.	► For specific information about the command you are using, see the "Restrictions" section of the command summary.
8	Mismatched Operands and Operator	The calculation specified by the boolean, variable or string expression cannot be done in the format of the variable or parameter to be loaded.	► Use N, AND, OR and XOR with boolean and integer operands only. ► Use LFT and RGT with integer operands only. ► Use SIN, COS, TAN, ATN, ETX, and LGN with floating point operands only.
9	Mathematical Overflow	The result of a variable expression in the program or motion block was outside of the allowed bounds of the variable.	► Reenter the variable expression in the program/motion block editor making sure that the operation will never go outside the allowed bounds of the variable. ► Consider using a double integer variable expression if you are working with very large numbers.
10	Mathematical Data Error	The result of a variable expression in the program or motion block cannot be stored as a number.	► Make sure that the SQR and LGN operators in the program/motion block never have negative operands. ► Make sure that a divide by zero operation will never occur in the program/motion block.

Bit	Fault Code Message	Possible Cause(s)	Possible Solution(s)
11	Value Out of Range	The value of a parameter obtained from a variable or expression was out of the range specified for the command in the program or motion block.	► Make sure that the variable or expression stays within the range of the parameter of the command. See the "Parameters" section of the command summary for the allowed range.
12	String Too Long	The result of a string variable operation in the program/ motion block was longer than 127 characters.	► Reenter the string variable operation in the program/motion block editor making sure that the result is not more than 127 characters.
13	Nonexistent Label	One of these commands was in the program with a label that does not exist in the program: GT, GS, IF...GT, IF...GS, WT...ON...GT, STB...GT.	► Reenter the command in the program editor making sure that the label exists in the program. ► Add the label number to the appropriate statement in the program.
14	Gosub Stack Underflow	The RET command was executed without there being a corresponding GS.	► Make sure that the program will execute a gosub the same number of times that it will execute a return. ► Check for program flow through a subroutine without a gosub call.
15	Gosub Stack Overflow	There were more than 32 nested gosubs in the program.	► Make sure that the program will execute a return the same number of times that it will execute a gosub. ► If the program leaves a subroutine without using a RET command, use the reset gosub stack command, RST, to reset the stack.
16	Invalid Variable Pointer	The pointer loaded in an integer variable in a program/ motion block was out of the range of variables available.	► Reenter the pointer in the program editor making sure that it is in the range of the type of variable accessed.
17	Invalid Motion	An attempt was made by the program/motion block to execute a run command while a home command was executing or vice versa; the combination of motion parameters define a motion that cannot be executed; or a motion command or motion block was executed when the controller was faulted.	► Make sure that the homing operation stops before executing a run command, or make sure that the motor is stopped before executing a home command. ► Make sure that the motion parameters define a motion that can be executed. For specific information about the parameters you are using, see the command summary. ► Make sure the controller is not faulted when executing a motion command or motion block.
21	Excessive Following Error	The following error, FE, was greater than the following error bound, FB.	► Make sure that the control constants are set up properly. ► Make sure that the position feedback wiring is correct. ► Make sure that the motor has sufficient torque.

Bit	Fault Code Message	Possible Cause(s)	Possible Solution(s)
22	Excessive Command Increment	Too many motions were simultaneously executed by the program.	► Make sure that the program does not execute too many motions simultaneously.
23	Position Register Overflow	The axis has moved past +/-2,000,000,000 pulses and position register wrap, PWE, is disabled.	► If the axis is to move constantly in one direction for long periods of time, PWE should be enabled. ► Make sure that the motion parameters define a motion that does not cause position register overflow. For specific information about the parameters you are using, see the command summary.
24	Resolver Feedback Lost	The resolver feedback became disconnected.	► Check the resolver feedback connection.
25	Motor Power Over-Voltage	The bus voltage was greater than 475 V.	► The clamp did not function correctly. Make sure that the wiring is correct.
26	Motor Power Clamp Excessive Duty Cycle - Under-Voltage	The internal clamp was operated past its rating of 50 W continuous, or the motor power is off.	► Try using an external clamp instead. ► Turn motor power on.
27	Motor Power Clamp Over-Current Fault	The external clamp resistance was less than 50 ohms.	► Make sure that the resistor value is equal to 50 ohms. ► Make sure that the resistor is correctly wired.
28	Motor Over-Current Fault	The controller was putting out excessive current through the motor leads.	► Check the wiring of the motor leads. ► Make sure that the motor leads are not shorted.
29	Motor Over-Temperature	The temperature sensor in the motor sensed the motor going over its maximum allowed temperature.	► Check for a broken wire in motor feedback cable. ► If motor is hot, it is improperly sized.
30	Controller Over-Temperature	The temperature of the controller heat sink was greater than 80 degrees Celsius.	► Check the controller for adequate air flow. A fan may be needed, or through-wall heat sink mounting can be used in order to allow for adequate air flow.
All Bits set to 0	Controller Functional	The controller is not faulted.	► Continue with normal operation.

Chart 3: Fault Input Messages

Bit	Fault Input Message	Possible Cause(s)
0	Resolver feedback lost input active	The resolver feedback is disconnected.
1	Motor power over-voltage input active	The bus voltage is greater than 475 V.
2	Motor power clamp or under-voltage input active	The internal clamp is on, or the motor power is off.
3	Motor power clamp over-current input active	The external clamp resistance is less than 50 ohms.
4	Motor over-current input active	The controller was putting out excessive current through the motor leads.
5	Motor over-temperature input active	The temperature sensor in the motor is sensing the motor temperature over its allowed maximum, or the motor feedback cable is not connected correctly.
6	Controller over-temperature input active	The temperature of the controller heat sink is greater than 80 degrees Celsius.
All bits set to 0	No fault input is active	There are no currently active fault inputs.

Chart 4: I/O Messages

Bit	I/O Message	Description
2	Axis channel A input active	Channel A of the encoder is active.
3	Axis channel B input active	Channel B of the encoder is active.
4	Auxiliary channel A input active	Channel A of the auxiliary encoder is active.
5	Auxiliary channel B input active	Channel B of the auxiliary encoder is active.
6	Auxiliary index input active	The index input of the auxiliary encoder is active.
7	Marker input active	The resolver of a resolver feedback unit is at 0 or the index input of an encoder feedback unit is active.
8	Home input active	The home input is active.
9	Forward overtravel input active	The forward overtravel input is active.
10	Reverse overtravel input active	The reverse overtravel input is active.
11	Enable input active	The enable input is active.
12	Capture input active	The position capture input is active.
13	Capture input edge	A positive edge was sensed on the position capture input.
15	OK output active	The OK output is active.
All bits set to 0	No I/O is active	None of the above I/O is active.

Chart 5: Program Status Messages

Bit	Program Status Message	Description
0	Program executing	The program specified is executing.
1	Program locked out	The program specified is being locked out by another program.
4	Invalid digit in string	The program specified has a floating point or integer variable expression with a string variable operand, and this string variable operand contained an invalid digit; or, the floating point or integer variable input by the IN command contained an invalid digit.
5	String value out of range	The program specified has a floating point or integer variable expression with a string variable operand, and this string variable operand contained a number out of the range of the variable; or, the floating point or integer variable input by the IN command was out of the range of the variable.
15	PROGRAM FAULT	The program specified caused the controller to fault.
Bit 0 set to 0	Program not executing	The program specified is not executing.

Chart 6: System Status Messages

Bit	System Status Message	Description
0	Program executing	One of the programs is executing.
1	Program locked out	One of the executing programs is being locked out by another program.
3	Motion block executing	One of the motion blocks is executing.
4	Key buffer empty	The key buffer contains no characters to be inputted by the GET or IN command.
5	Transmit buffer full	The transmit buffer of the controller is full.
12	I/O FAULT	A digital output fault has occurred. See the "Digital Output Fault" message of the "Fault Code Message" section for more information.
13	AXIS FAULT	A fault specific to the axis has occurred.
14	SYSTEM FAULT	A fault has occurred. This could be any fault possible in the controller.
15	MEMORY FAULT	A memory fault has occurred due to the user program memory not checksumming.
Bit 0 set to 0	No program executing	None of the programs are executing.

Chart 7: Axis Status Messages

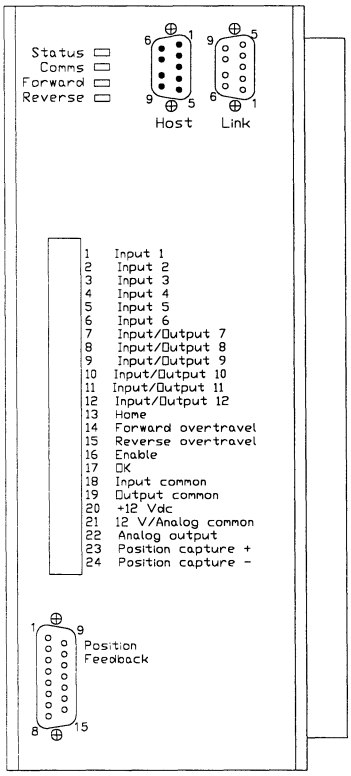
Bit	Axis Status Message	Description
0	Motion generator enabled	The motion generator is enabled.
1	Ratio input enabled	The ratio input, RIE, is enabled.
2	Phase locked loop enabled	The phase locked loop, PHE, is enabled.
3	Motion block executing	One of the motion blocks is executing.
4	Motion suspended	The motion of the axis has been suspended.
5	Phase error past bound	The phase error is past the phase error bound, PHB.
6	Axis accel/decel	The axis is either accelerating or decelerating.
7	Axis direction forward	The axis is moving or has last moved in the forward direction.
8	Axis in position	The axis is stopped and within the position band, PB, of the command position, CP.
9	Axis at torque limit	The torque limit enable, TLE, is enabled, and the axis is at the torque limit set by the torque limit current, TC.
10	Axis at overtravel	The axis is either at a hardware overtravel input or a software overtravel limit.
11	Axis at software overtravel	The axis is at a software overtravel limit.
13	AXIS FAULT	A fault specific to the axis has occurred.
Bit 7 set to 0	Axis direction reverse	The axis is moving or has last moved in the reverse direction.

Appendix D

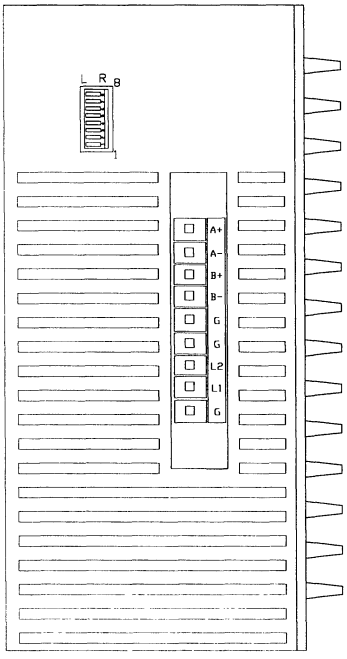
User Connections

IMC - 105X - 1 - C and IMC - 105E - 1 - C

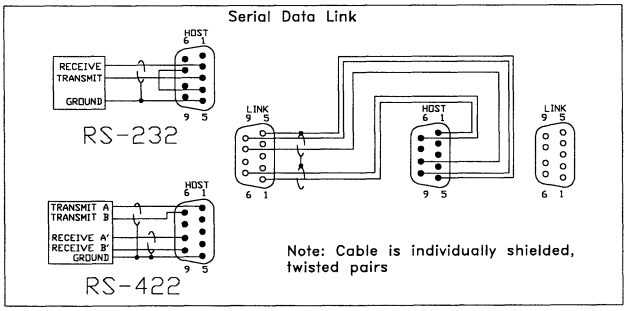
USER CONNECTIONS AND SWITCH SETTINGS

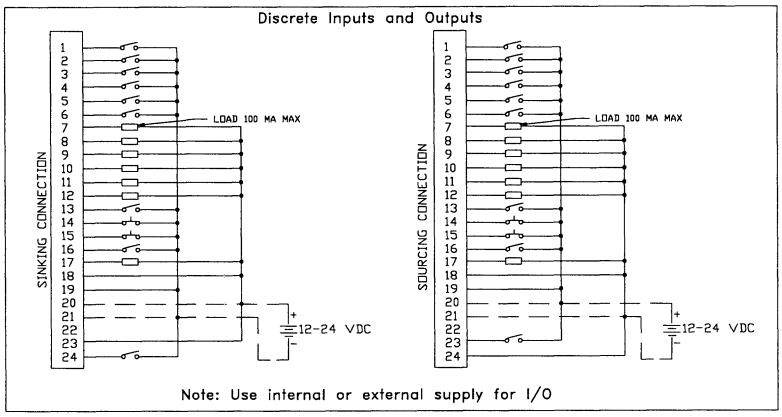


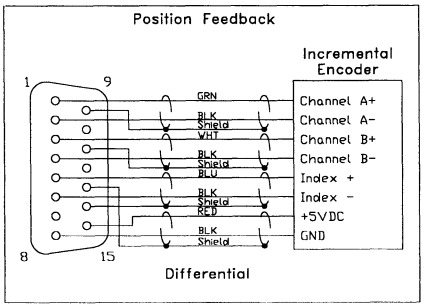
FRONT VIEW

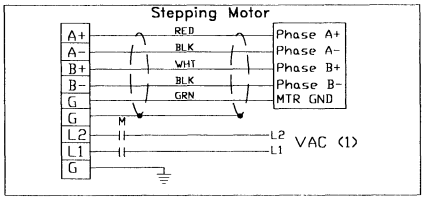


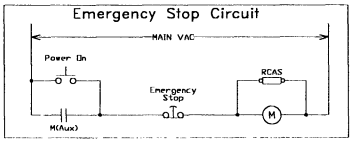
BOTTOM VIEW











DIP Switch Positions (2)

Unit Address	1	2	3	4	5
0	R	R	R	R	R
1	L	R	R	R	R
2	R	L	R	R	R
3	L	L	R	R	R
4	R	R	L	R	R
5	L	R	L	R	R
6	R	L	L	R	R
7	L	L	L	R	R
8	R	R	R	L	R
9	L	R	R	L	R
A	R	L	R	L	R
B	L	L	R	L	R
C	R	R	L	L	R
D	L	R	L	L	R
E	R	L	L	L	R
F	L	L	L	L	R
G	R	R	R	R	L
H	L	R	R	R	L
I	R	L	R	R	L
J	L	L	R	R	L
K	R	R	L	R	L
L	L	R	L	R	L
M	R	L	L	R	L
N	L	L	L	R	L
O	R	R	R	L	L
P	L	R	R	L	L
Q	R	L	R	L	L
R	L	L	R	L	L
S	R	R	L	L	L
T	L	R	L	L	L
U	R	L	L	L	L
V	L	L	L	L	L

Serial Baud Rate	6	7
1200	R	R
9600	L	R
19200	R	L
38400	L	L

Switch 8 is Reserved

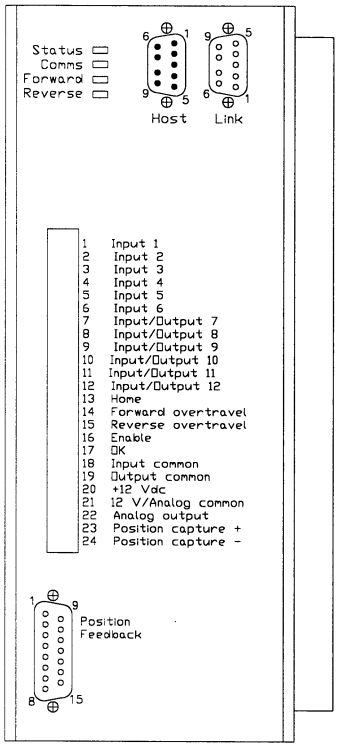
REMARKS:

(1) Input power 90 to 130 VAC
1 phase 50-440 Hz @ 10 Amps

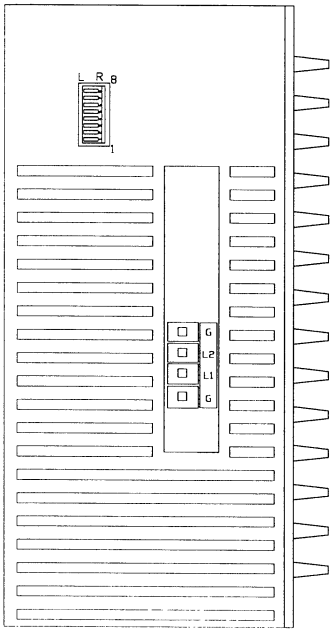
(2) Must turn off power before changing settings,
R= right (closed)
L= left (open)

IMC - 200X - X - C and IMC - 200E - X - C

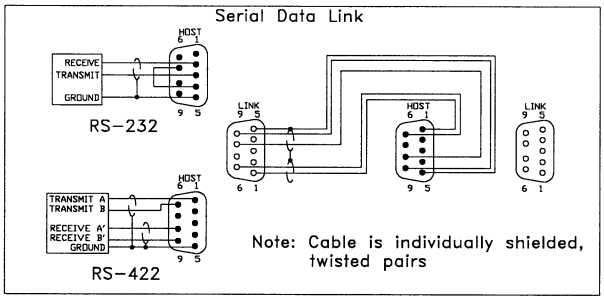
USER CONNECTIONS AND SWITCH SETTINGS

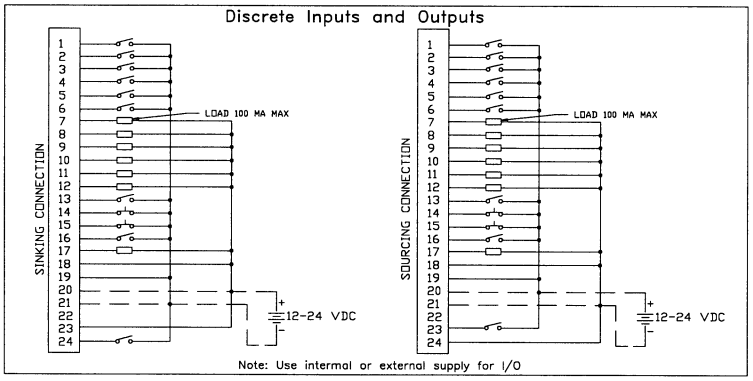


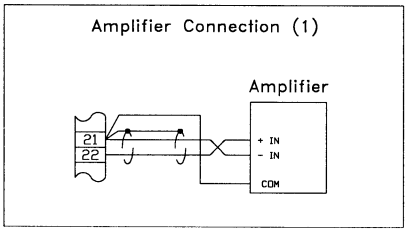
FRONT VIEW

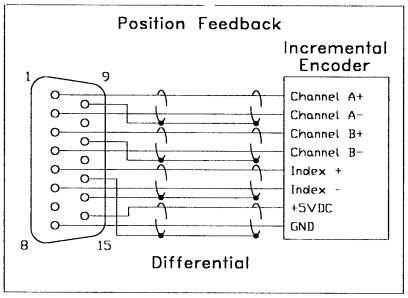


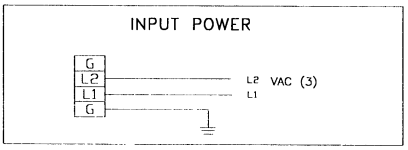
BOTTOM VIEW











DIP Switch Positions (2)

Unit Address	1	2	3	4	5
0	R	R	R	R	R
1	L	R	R	R	R
2	R	L	R	R	R
3	L	L	R	R	R
4	R	R	L	R	R
5	L	R	L	R	R
6	R	L	L	R	R
7	L	L	L	R	R
8	R	R	R	L	R
9	L	R	R	L	R
A	R	L	R	L	R
B	L	L	R	L	R
C	R	R	L	L	R
D	L	R	L	L	R
E	R	L	L	L	R
F	L	L	L	L	R
G	R	R	R	R	L
H	L	R	R	R	L
I	R	L	R	R	L
J	L	L	R	R	L
K	R	R	L	R	L
L	L	R	L	R	L
M	R	L	L	R	L
N	L	L	L	R	L
O	R	R	R	L	L
P	L	R	R	L	L
Q	R	L	R	L	L
R	L	L	R	L	L
S	R	R	L	L	L
T	L	R	L	L	L
U	R	L	L	L	L
V	L	L	L	L	L

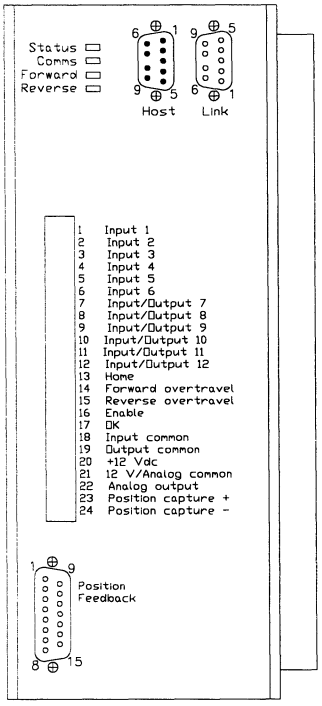
Serial Baud Rate	6	7
1200	R	R
9600	L	R
19200	R	L
38400	L	L

Switch 8 is Reserved

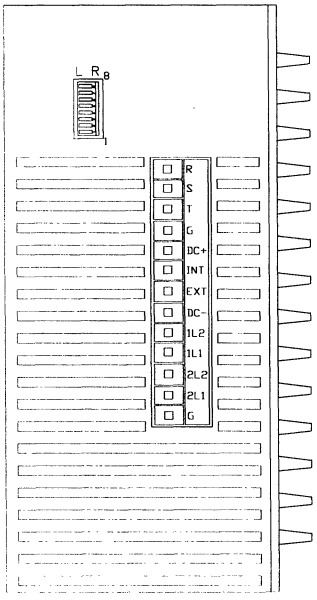
REMARKS:

(1) Connections may depend on the type of amplifier, consult amplifier manufacturer for more information.
(2) Must turn off power before changing settings,
R= right (closed)
L= left (open)
(3) Input power 90 to 250 VAC
1 phase 50-440 Hz @ 1 Amp

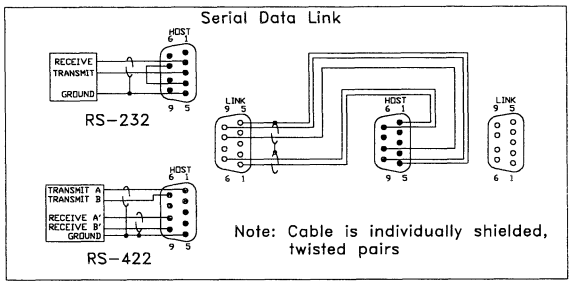
USER CONNECTIONS AND SWITCH SETTINGS

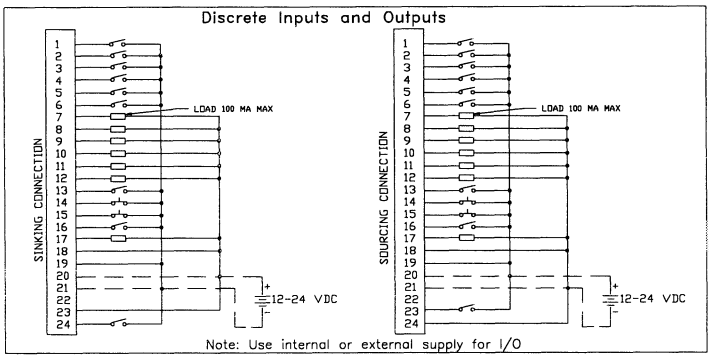


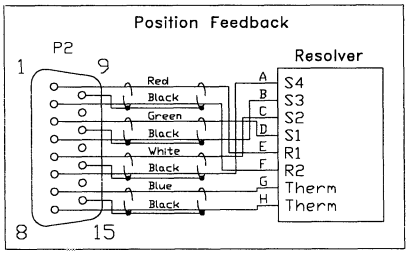
FRONT VIEW

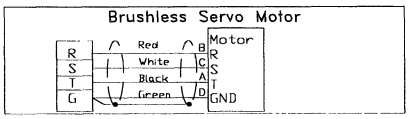


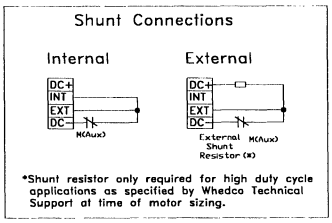
BOTTOM VIEW

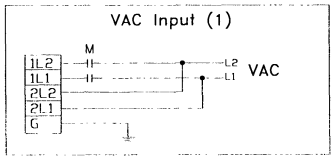


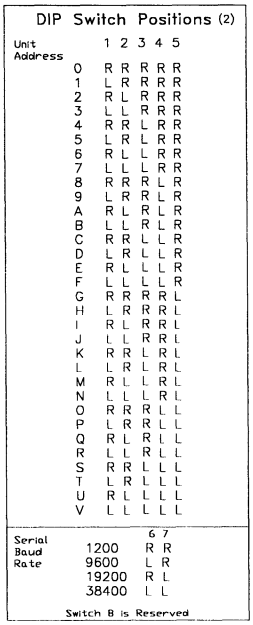


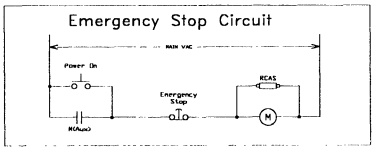










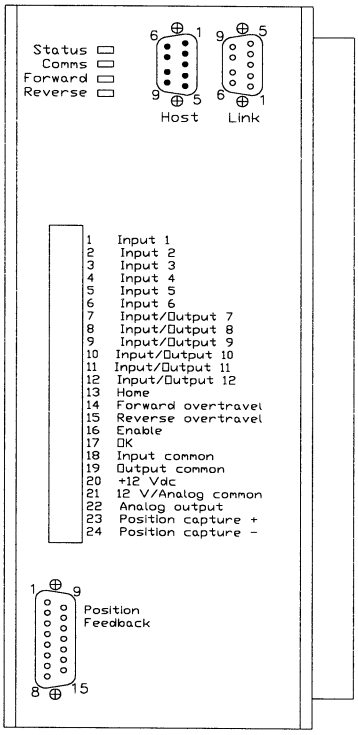


REMARKS:
(1) Input power 90 to 250 VAC
1 phase 50-440 Hz @ 15 Amps
(2) Must turn off power before changing settings,
R= right (closed)
L= left (open)

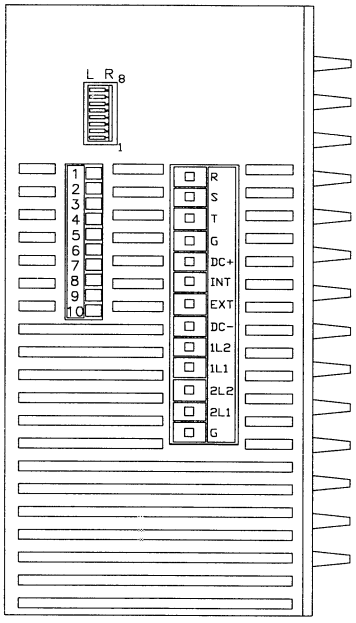
Additional *Connections* for *Enhanced* Options

Models

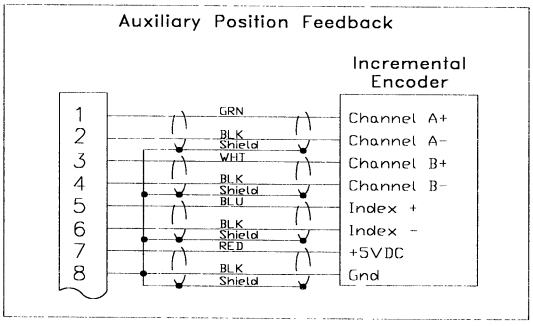
IMC-105E-1-C IMC-200E-X-C IMC-313E-X-C IMC-316E-X-C

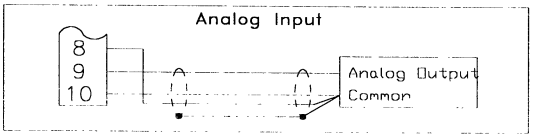


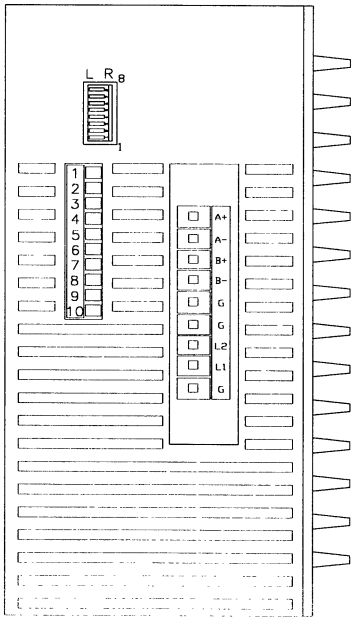
FRONT VIEW



BOTTOM VIEW



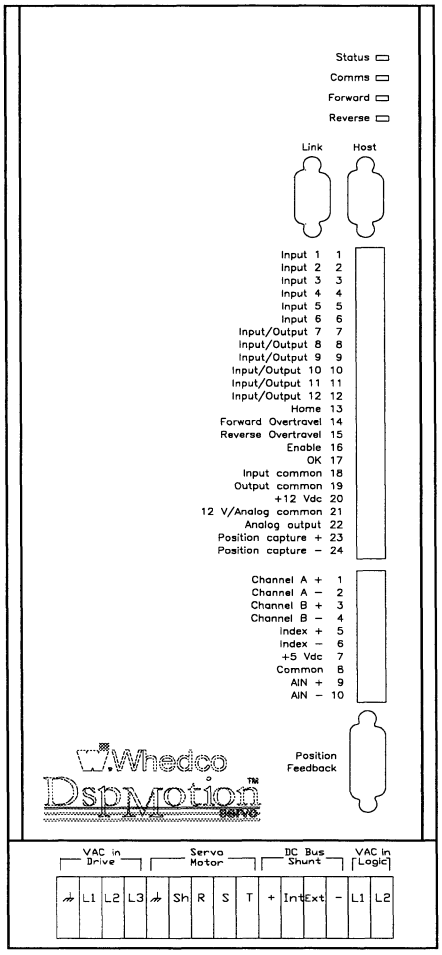




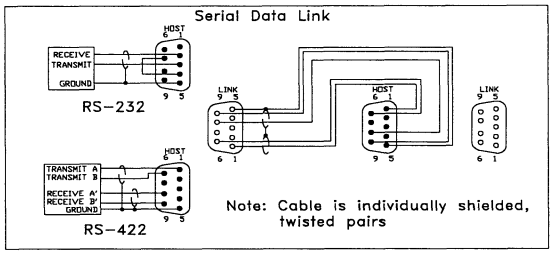
BOTTOM VIEW

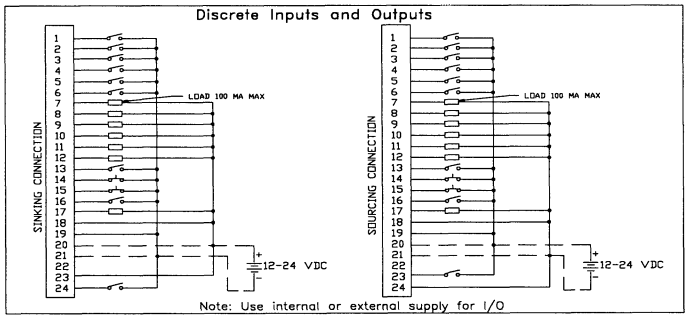
IMC-31CE-X-C and IMC-31PE-X-C

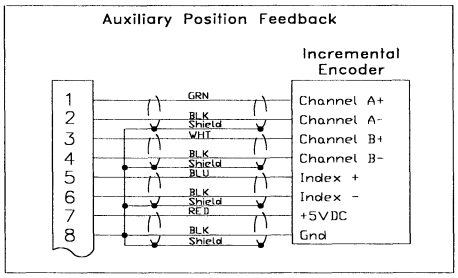
USER CONNECTIONS AND SWITCH SETTINGS

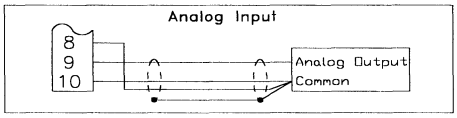


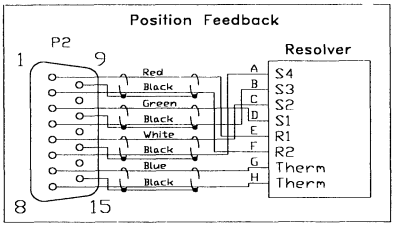
Front View





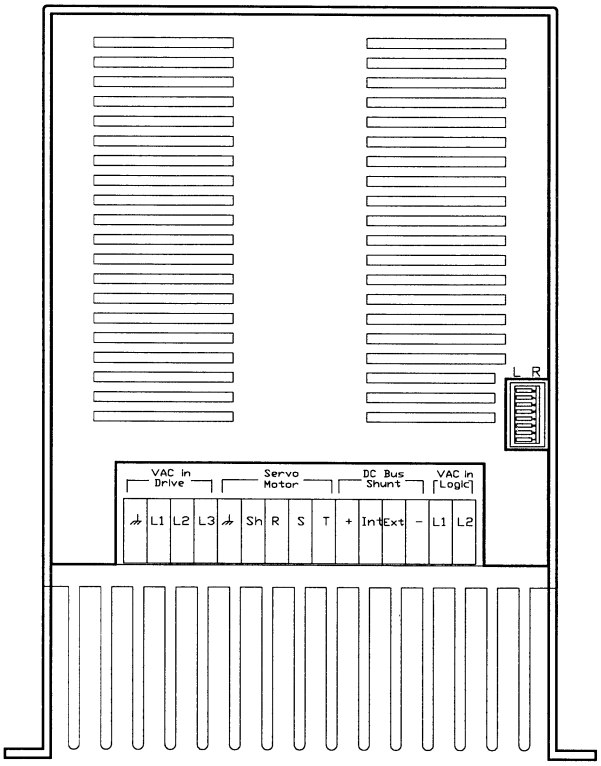






IMC-31CE-X-C and IMC-31PE-X-C

USER CONNECTIONS AND SWITCH SETTINGS



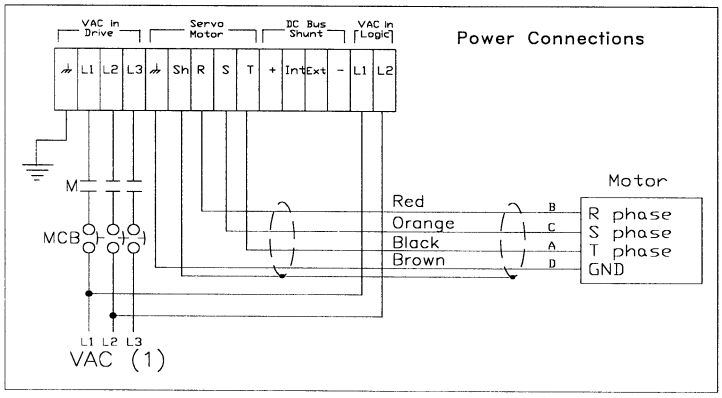
BOTTOM VIEW

DIP Switch Positions (2)

Unit Address	1	2	3	4	5
0	L	L	L	L	L
1	R	L	L	L	L
2	L	R	L	L	L
3	R	R	L	L	L
4	L	L	R	L	L
5	R	L	R	L	L
6	L	R	R	L	L
7	R	R	R	L	L
8	L	L	L	R	L
9	R	L	L	R	L
A	L	R	L	R	L
B	R	R	L	R	L
C	L	L	R	R	L
D	R	L	R	R	L
E	L	R	R	R	L
F	R	R	R	R	L
G	L	L	L	L	R
H	R	L	L	L	R
I	L	R	L	L	R
J	R	R	L	L	R
K	L	L	R	L	R
L	R	L	R	L	R
M	L	R	R	L	R
N	R	R	R	L	R
O	L	L	L	R	R
P	R	L	L	R	R
Q	L	R	L	R	R
R	R	R	L	R	R
S	L	L	R	R	R
T	R	L	R	R	R
U	L	R	R	R	R
V	R	R	R	R	R

Serial Baud Rate	6	7
1200	L	L
9600	R	L
19200	L	R
38400	L	L

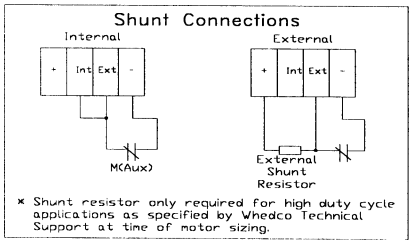
Switch 8 is Reserved

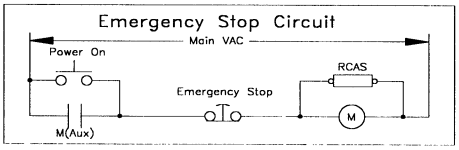


REMARKS:

(1) Input power 180 to 250 VAC
3 phase 50–440 Hz
@ 20 Amps

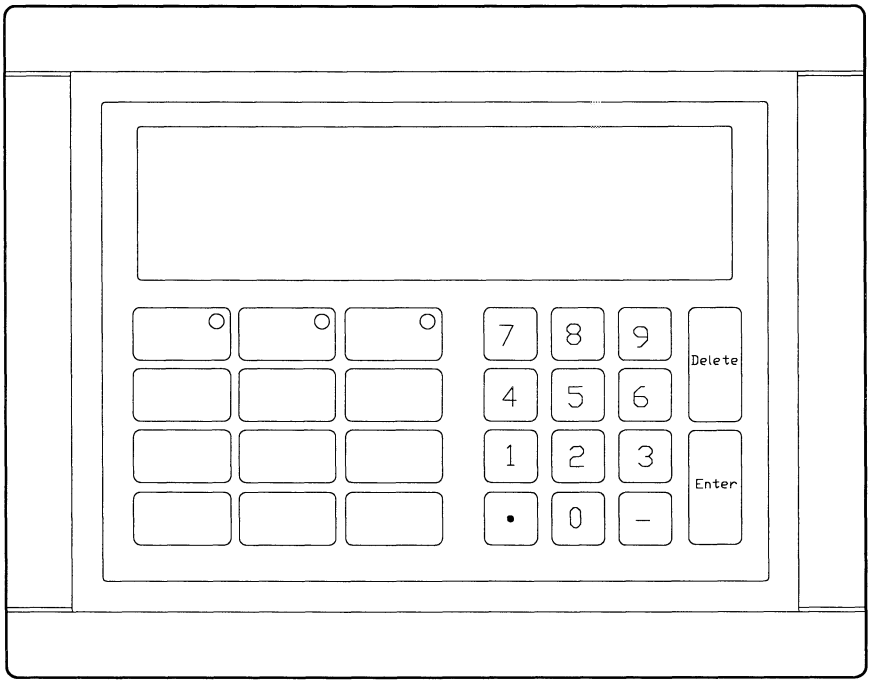
(2) Must turn off power before changing settings,
R= right (closed)
L= left (open)



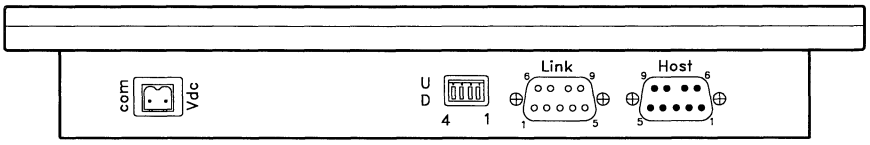


OIP - DSP1 - C

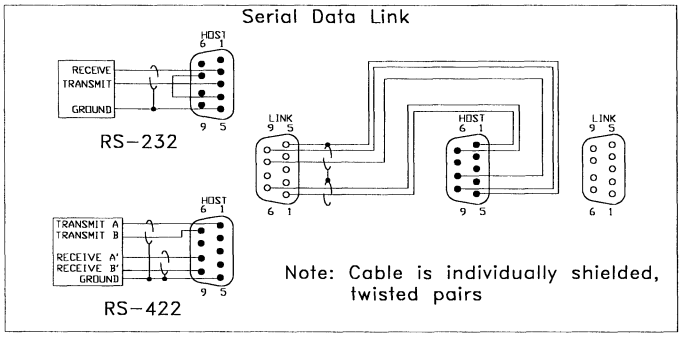
USER CONNECTIONS AND SWITCH SETTINGS



FRONT VIEW



BOTTOM VIEW

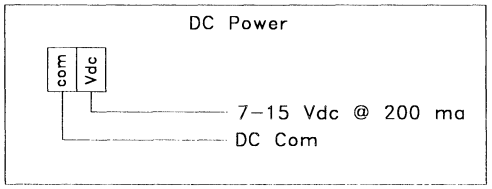


DIP Switch Positions(1)

serial baud rate	1	2
1200	U	U
9600	D	U
19200	U	D
38400	D	D

Switches 3 and 4 are reserved

Remarks:
(1) Must turn off power before changing settings,
U= up (closed)
D= down (open)



Appendix E

Specifications

Motion Parameters

Description	Minimum	Maximum	Resolution	Units
Absolute Position Register	-2,000,000,000	+2,000,000,000	1	pulses
Absolute Move Distance	-2,000,000,000	+2,000,000,000	1	pulses
Incremental Move Distance	-2,000,000,000	+2,000,000,000	1	pulses
Offset Move Position	-2,000,000,000	+2,000,000,000	1	pulses
Acceleration/Deceleration	100	1,000,000,000	2	pulses/second^2
Velocity	1	16,000,000	1	pulses/second
Acceleration/Deceleration Percent	1	99	1	percent
Move Time	0.01	650	0.01	seconds
Jerk Percent	0	100	1	percent
Unit Ratio[(a)]	1	1,000,000	1	pulses/unit
Electronic Gear Ratio	+/- 1: 10,000	+/- 10,000 : 1	1	pulses : pulses
Phase-Locked Loop Length	500	64,000	1	pulses

(a) The unit ratio is used to convert to engineering units. To determine the range of permissible values for each parameter in engineering units, divide the parameter minimum and maximums by the number assigned to the unit ratio.

Memory

Type	Size/Range	Number
Programs	32 Kbytes	4
Motion Blocks	32 Kbytes	100
Integer[(a)]	+/-2,147,483,647	4,096
Floating Point Variables[(a) (b)]	+/-1.7X10$^{+/-38}$	2,048
Boolean Variables	1-bit	256
String Variables	127 characters	16
Countdown Timers	2,000,000.000 seconds	8
Total User Program Storage	32 Kbytes BBRAM or EPROM	
Total Variable Storage	16.5 Kbytes BBRAM	

(a) Integer and Floating Point Variable space is shared. There can be a maximum of 4096 integer variables or 2048 floating point variables, but not both. Integer and Floating Point Variables can be mixed with memory space shared (for example, if 1024 Floating Point Variables are used, 2048 integer variables are possible).

Timing

Motion Loop Update Rate	122 microseconds
Position Capture Update Rate	400 nanoseconds
Countdown Timer Update Rate	1 millisecond
Analog Output Update Rate	122 microseconds
Analog Input Bandwidth	1 kHz

Mathematical Operands

add, subtract, multiply, divide, square root, absolute value, exponential, natural log, sine, cosine, tangent, arc tangent (trigonometric functions are in degrees)

Communication

Format	RS - 232	RS - 422
Maximum Addressable Units	32[(a)]	31[(b)]
Maximum Distance from Host to Unit	50 feet	1000 feet
Maximum Length of Serial Data Link	1000 feet	1000 feet
Baud Rate	1200, 9600, 19200 or 38400	

(a) Operator Interface counts as a unit
(b) Operator Interface does not count as a unit

Discrete Inputs and Outputs

Inputs

Operating Range	12-24 Vdc, 30 Vdc maximum
Maximum Off Input Voltage	4 Vdc
Minimum On Input Voltage	10 Vdc
Load	2K Ohms
Interface Format	source/sink user configurable

Outputs

Operating Range	12-24 Vdc, 30 Vdc maximum
Maximum On Resistance	35 Ohms
Maximum Load Current	100 mA
Maximum Off Leakage Current	200 nA
Interface Format	source/sink user configurable

Analog Inputs and Outputs

Inputs

Model	Standard	Enhanced
Number	0	1
Operating Range	+/- 10 Vdc	
Resolution	12 bits	
Input Impedance	50K Ohms	

Outputs

Model	Stepper	Servo with User-Supplied Drive	Servo with Integral Drive
Number	1	1	1
Parameter	user programmable, or velocity, current, or following error	control output to external drive	user programmable, or velocity, current, or following error
Operating Range	+/- 10 Vdc		
Resolution	12 bits		
Current	5 mA		

Position and Velocity Feedback

Incremental Encoder

Controller Type	Standard Model: Stepper & Servo with User-Supplied Drive	Standard Model: Servo with Integral Drive	Enhanced Model: Stepper & Servo with User-Supplied Drive	Enhanced Model: Servo with Integral Drive
Number	1	0	2	1
Functions Supported	- primary position feedback	- n/a	- primary position feedback - electronic gearing - phase-locked-loop - secondary position feedback	- electronic gearing - phase-locked-loop - secondary position feedback
Note: All servo units (IMC - 31☐☐ - ☐ - C) also have a resolver for primary position feedback and motor commutation.				

Resolver

Controller Type	Stepper with Integral Drive & Servo with User-Supplied Drive	Servo with Integral Drive
Number	none	1
Function		position & velocity feedback
Resolution		4096 pulses per revolution
Maximum Speed	15,000 rpm	
Type	control transmitter	
Phase Shift	+/- 5.0 degrees @ 5 kHz	
Null Voltage	< 20 mV @ 5 kHz	
Transformation Ratio	0.50 to 2.0	

DC Power Supplies

Controller Type	Stepper with Integral Drive & Servo with User-Supplied Drive	Servo with Integral Drive
+5 volts	0.5 Amps	n/a
+12 volts	0.5 Amps	0.5 Amps

Environmental

Power Level	Stepper & Servo Controllers up to 2 HP (Enclosure Size A)	Servo Controllers from 5 to 10 HP (Enclosure Size B)
Operating Temperature, Free Air Ambient[a] [c]	0 to 50 degrees C	0 to 40 degrees C
Storage and Shipping Temperature[b]	- 40 to 80 degrees C	
Relative Humidity	5 to 95% noncondensing	
Enclosure	NEMA 1 ▪ CSA 1 ▪ IP 20	

(a) assumes heatsink orientation is vertical
(b) contents of user-programmed BBRAM may be lost if temperature drops below O°C
(c) fan cooling is required for duty cycles greater than 50%

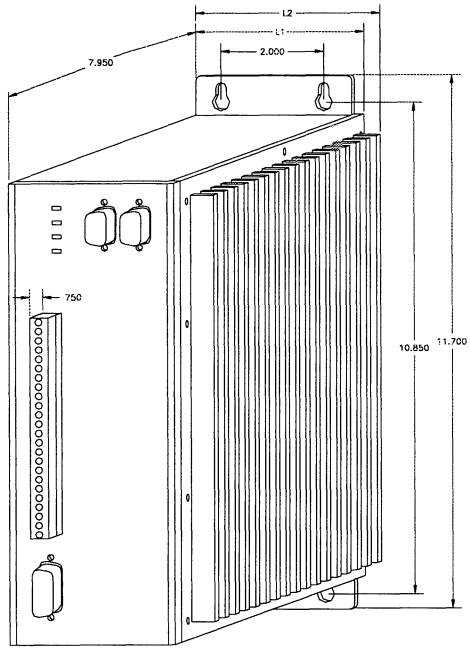
Mechanical

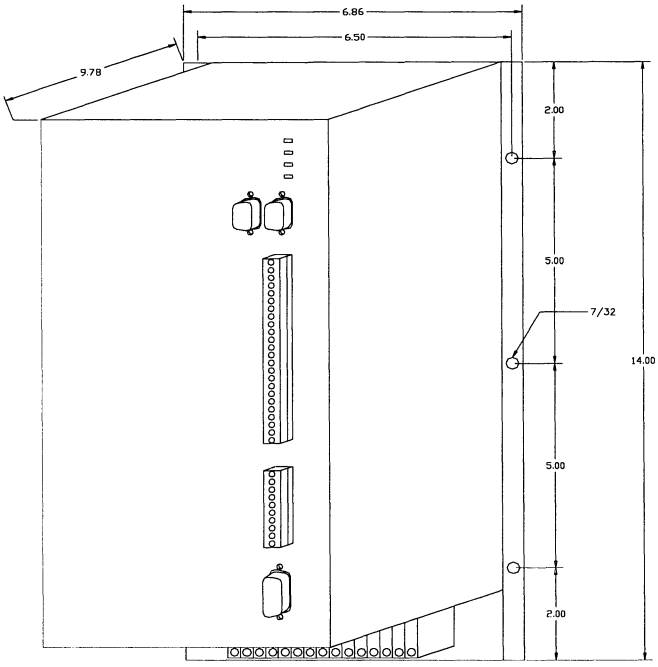
Dimensions are shown in inches, weight in pounds

Enclosure Size A		
Model	Width	Weight
IMC -105☐- 1 - C	L2	8
IMC - 200☐ - X - C	L1	7
IMC - 313☐ - X - C	L2	8
IMC - 316☐ - X - C	L2	8

L1 = 3.720

L2 = 4.200





Enclosure Size B
Model
IMC - 31CE - 2 - C
IMC - 31PE - 2 - C

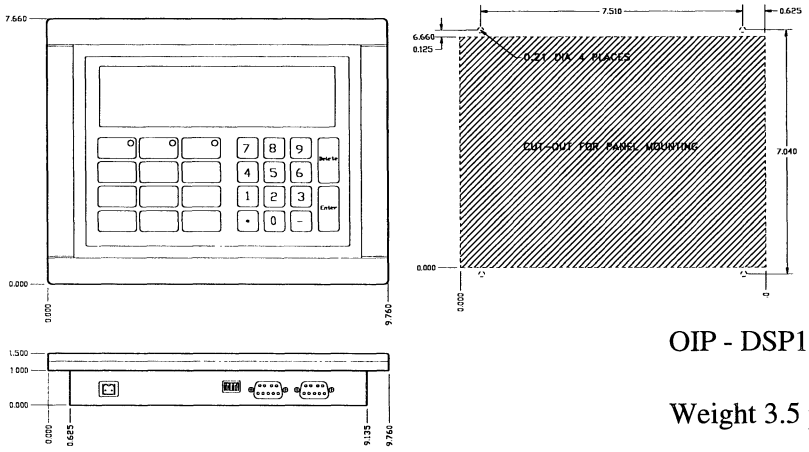
Weight = 23 pounds

Mounting dimensions, center to center = 6.50

Total width = 6.86

Depth = 9.78

Height = 14.00



OIP - DSP1 - C

Weight 3.5 pounds approximate

Appendix F

IMC-B/IMC-C Comparison

Appendix F: IMC-B/IMC-C Comparison

Section 1: Conversion from IMC-B to IMC-C

In order to convert IMC-B profiles to IMC-C programs/motion blocks, we must first recognize that the execution of profiles and the execution of programs and motion blocks is somewhat different. Programs cannot execute complex motion profiles, such as multiple speed moves. All complex motions (and other types of motions) are accomplished by motion blocks. In addition, many of the motion commands implemented in the IMC-B unit have been done away with in favor of a simplified motion command set. All things such as setting the position registers and waiting for trigger inputs to occur are taken care of by other commands. So, many IMC-B motion commands can be implemented by a series of IMC-C commands.

Next, many of the parameter units are different. Consult the command summaries of both models so that correct conversions can be made. Another thing to be recognized is that the assignment symbols "=" and ":" have been added to many commands. Again, refer to the command summary for more information.

The next section compares the commands from the IMC-B to the IMC-C in order to give an idea of how to convert IMC-B profiles to IMC-C programs/motion blocks. Then, in the section afterwards, an example is given that shows how the two units compare from a programming standpoint.

Section 2: Command Comparison

Type	IMC-B	IMC-C	Remarks
System	CB	CLM	The CB command faults the controller; the CLM command does not.
System	EB		Not available
System	WB	RSF	Functionally equivalent
System	PE		Not available; the IMC-C can be programmed to execute a motion using the digital inputs. See Appendix F, Section 3.
System	ML		Not available
System	MS		Not available
System	WP		Not available; the IMC-C automatically executes program 4 when a controller fault occurs.
Axis	PO	FO	Functionally equivalent
Axis	NO	RO	Functionally equivalent
Axis	OT		Not available
Axis	WR	PWE	Functionally equivalent
Axis	DB	DB	Functionally equivalent
Axis	PB	PB	Functionally equivalent
Axis	LD	FR	FR does not set following error bound. See FB in the command summary.
Axis	GN	KP	Functionally similar; see command summary for more information.
Axis	IB	KI	Functionally similar; see command summary for more information.
Axis	ZR	KD	Functionally similar; see command summary for more information.
Axis	FF		Not required; this is set automatically by the IMC-C.
Axis	PL		Not available
Axis	GR		Not available
Axis	PS	SC	SC allows the user to set the reduction in current from 0% to 100%.
Axis	SS		Not available

Type	IMC-B	IMC-C	Remarks
Axis	ER		Not used; see FR in the command summary.
Axis	DF		Not available; the user can set the deadband to be greater than the following error bound, FB.
Axis	PC		Not available; when using encoder feedback, position correction is always enabled. If position correction is not desired, set the deadband constant, DB, to its maximum value.
Axis	CT	CT	Functionally equivalent
Axis	BL	BL	Functionally equivalent
Axis	RN		Not available
Program	DE	DE	The IMC-B command is for profiles; the IMC-C command is for programs.
Program	!	!	Functionally equivalent
Program	X	X	The IMC-C command has extra capabilities; see command summary for more information.
Program	DEL	DEL	Functionally equivalent
Program	ED	ED	The IMC-B command is for profiles; the IMC-C command is for programs.
Program	RM		Not available; repeating a move can be accomplished by using a loop in an IMC-C program. See "Programming" for more information.
Program	GS	GS	The IMC-B command is for profiles; the IMC-C command is for labels.
Program	GT	GT	The IMC-B command is for profiles; the IMC-C command is for labels.
Program	IF...GT	IF...GT	The IMC-B command is for profiles; the IMC-C command is for labels.
Program	IF...GS	IF...GS	The IMC-B command is for profiles; the IMC-C command is for labels.
Program	MW	WT IP	Functionally equivalent
Program	TW	WT TL	Functionally equivalent
Program	AW	WT N SRA6	Functionally equivalent
Program	EX	EX	The IMC-B command is for profiles; the IMC-C command is for programs.
Register	AM	AM	Functionally equivalent
Register	IM	IM	Functionally equivalent
Register	AC	AC	Functionally equivalent
Register	DC	DC	Functionally equivalent
Register	SP	VL	Functionally equivalent
Register	BS		Not available; see motion commands.
Register	DT		Not available; see motion commands.
Register	AP		Not available; see motion commands.
Register	IP		Not available; see motion commands.
Register	LR		Not available; see RM.
Register	PIZ	SAP:0	Functionally equivalent
Register	PIA	SAP:*n*	Functionally equivalent

Type	IMC-B	IMC-C	Remarks
Motion	HT	HT	Functionally equivalent
Motion	ST	ST	Functionally equivalent
Motion	RAN	RAN	Functionally equivalent
Motion	RAI	AM=*n*, RAN	Functionally equivalent
Motion	RAD	STM*t*=*n*, WTTM*t*, RAN	Functionally equivalent; *n* should be set to the value from the DT parameter.
Motion	RAT	WT DI*n*, RAN	Functionally equivalent; digital input *n* of the IMC-C is used as the trigger 1 input.
Motion	RAU	WT DI*n*, RAN	Functionally equivalent; digital input *n* of the IMC-C is used as the trigger 2 input.
Motion	RkS		Use a motion block to define the same motion.
Motion	RkN	RIN	Functionally equivalent
Motion	RkI	IM=*n*, RIN	Functionally equivalent
Motion	RkD	STM*t*=*n*, WTTM*t*, RIN	Functionally equivalent; *n* should be set to the value from the DT parameter.
Motion	RkT	WT DI*n*, RIN	Functionally equivalent; digital input *n* of the IMC-C is used as the trigger 1 input.
Motion	RkU	WT DI*n*, RIN	Functionally equivalent; digital input *n* of the IMC-C is used as the trigger 2 input.
Motion	SkN	RkN	Functionally equivalent
Motion	SkD	STM*t*=*n*, RkN, WTTM*t*, ST	Functionally equivalent; *n* should be set to the value from the DT parameter.
Motion	SkE	STM*t*=*n*, WTTM*t*, RkN	Functionally equivalent; *n* should be set to the value from the DT parameter.
Motion	SkU	WT DI*n*, RkN	Functionally equivalent; digital input *n* of the IMC-C is used as the trigger 1 input.
Motion	SkV	WT DI*n*, RkN	Functionally equivalent; digital input *n* of the IMC-C is used as the trigger 2 input.
Motion	SkW	WT DI*n*, RkN, WT DI*m*, ST	Functionally equivalent; digital input *n* of the IMC-C is used as the trigger 1 input, and digital input *m* of the IMC-C is used as the trigger 2 input.
Motion	SkT	RkN, WT DI*n*, ST	Functionally equivalent; digital input *n* of the IMC-C is used as the trigger 2 input.
Motion	SkA		Not available; use a motion block to define the same motion.
Motion	SkI		Not available; use a program to define the same motion.
Motion	SkL		Not available; use a program to define the same motion.
Motion	SkG		Not available; use a program to define the same motion.
Motion	SkS		Not available; use a program to define the same motion.
Motion	SkJ	RkN, WT IO7 AND IO8, ST	Functionally equivalent
Motion	SkH	RkN, WT IO8, ST	Functionally equivalent

Type	IMC-B	IMC-C	Remarks
Motion	SkO	RkN, WT N IO8, ST	Functionally equivalent
Motion	SFX	RFN, WT N DI*n*, ST	Digital input *n* of the IMC-C is used as the trigger 1 input.
Motion	SRX	RRN, WT N DI*n*, ST	Digital input *n* of the IMC-C is used as the trigger 2 input.
Motion	TM	TC=*n*, TLE=1, RkN	Functionally equivalent; the sign of the torque move parameter determines the use of RFN or RRN.
Motion	PkH	HkH, WT IP, SAP:0	Functionally equivalent
Motion	PkI	HkH, WT IP, SAP:*n*	Functionally equivalent; *n* is the value of the AP parameter.
Motion	PkJ	HkM, WT IP, SAP:0	Functionally equivalent
Motion	PkK	HkM, WT IP, SAP:*n*	Functionally equivalent; *n* is the value of the AP parameter.
Motion	PkB	HkH, WT IP, HkM, WT IP, SAP:0	Functionally equivalent
Motion	PkC	HkH, WT IP, HkM, WT IP, SAP:*n*	Functionally equivalent; *n* is the value of the AP parameter.
Motion	SI	SI	Functionally equivalent
In/Out	PTpxxB	SPA*n*B; SPB*n*B	Functionally equivalent; see command summary for parameter ranges.
In/Out	PTpxxE	SPA*n*E; SPB*n*E	Functionally equivalent; see command summary for parameter ranges.
In/Out	PTpxxF	SPA*n*F; SPB*n*F	Functionally equivalent; see command summary for parameter ranges.
In/Out	STp	STD11, STD12	Functionally equivalent
In/Out	RSp	RSD11, RSD12	Functionally equivalent
In/Out	"..."	PRT	Functionally equivalent; DSE of the IMC-C must be set to 0 so that the string will be sent to the terminal.
Diagnostic	DG	DGE	Functionally equivalent
Diagnostic	RS	RSRS, RSRA	Functionally similar; see command summary for more information.
Diagnostic	FC	RFC	Functionally similar; see command summary for more information.
Diagnostic	RC	RCP	Functionally equivalent
Diagnostic	RP	RAP	Functionally equivalent
Diagnostic	FE	RFE	Functionally equivalent
Diagnostic	RE		Not available
Diagnostic	RT		Not available
Diagnostic	FR	RRR	Functionally equivalent
Diagnostic	VL	RAV	Functionally similar; see command summary for more information.

Section 3: Programming Example

IMC Model B

DE1	(define profile 1)
SP5000	(load speed of 500 rpm)
AC25000	(load acceleration of 25,000 rpm/sec)
SFN	(slew axis forward)
ED	(end profile 1 and exit profile editor)
DE2	(define profile 2)
SP5000	(load speed of 500 rpm)
AC25000	(load acceleration of 25,000 rpm/sec)
SRN	(slew axis reverse)
ED	(end profile 2 and exit profile editor)
DE3	(define profile 3)
SP5000	(load speed of 500 rpm)
AC25000	(load acceleration of 25,000 rpm/sec)
AM81920	(load absolute move position of 81,920 pulses = 20 revs * 4,096 pulses/rev)
RAN	(run axis to absolute move position)
ED	(end profile 3 and exit profile editor)
DE4	(define profile 4)
SP5000	(load speed of 500 rpm)
AC25000	(load acceleration of 25,000 rpm/sec)
IM49152	(load incremental move distance of 49,152 pulses = 12 revs * 4,096 pulses/rev)
RFN	(run axis for incremental move distance)
ED	(end profile 4 and exit profile editor)
DE5	(define profile 5)
PFJ	(slew axis forward until index input, stop motor, set absolute position register equal to 0)
ED	(end profile 5 and exit profile editor)
DE6	(define profile 6)
ST	(decelerate axis to stop motor)
ED	(end profile 6 and exit profile editor)
PE1	(enable profile execution from parallel inputs)

Once the IMC-3 model B is powered up, the controller will be faulted. The controller will reset all faults once the enable input changes from inactive to active. Once this happens, the controller will be ready to handle parallel profile inputs 1 through 6, which will execute profiles 1 through 6 once made active. These profiles will execute a specified motion.

IMC Model C

```
     AUR=4096              (load axis unit ratio of 4,096 pulses/unit, therefore units=revolutions)

     DE1                   (define program 1)
     VL=500/60             (load velocity of 500 rpm/(60 sec/min) = 8.3333 rev/sec)
     AC=25000/60           (load acceleration of (25,000 rpm/sec)/(60 sec/min) = 416.6667 rev/sec^2)
     AM=20                 (load absolute move position of 20 revs = 81,920 pulses/(4,096 pulses/rev))
     IM=12                 (load incremental move position of 12 revs = 49,152 pulses/(4,096 pulses/rev))
1    IF EG1 GS10           (conditionally gosub 10 if positive edge sensitive digital input 1 becomes active)
     IF EG2 GS20           (conditionally gosub 20 if positive edge sensitive digital input 2 becomes active)
     IF EG3 GS30           (conditionally gosub 30 if positive edge sensitive digital input 3 becomes active)
     IF EG4 GS40           (conditionally gosub 40 if positive edge sensitive digital input 4 becomes active)
     IF EG5 AND IP GS50    (conditionally gosub 50 if positive edge sensitive digital input 5 becomes active
                            and motor is in position)
     IF EG6 GS60           (conditionally gosub 60 if positive edge sensitive digital input 6 becomes active)
     GT1                   (unconditionally goto 1)
10   RFN                   (run forward)
     RET                   (return from gosub)
20   RRN                   (run reverse)
     RET                   (return from gosub)
30   RAN                   (run to absolute move position)
     RET                   (return from gosub)
40   RIN                   (run to incremental move position)
     RET                   (return from gosub)
50   HFM                   (home forward to marker)
     WT IP                 (wait for motor to be in position)
     SAP:0                 (set absolute position register equal to 0)
     RET                   (return from gosub)
60   ST                    (stop all motion)
     RET                   (return from gosub)
     ED                    (end program 1 and exit program editor)

     DE4                   (define program 4)
     KL1                   (kill program 1)
     WT N IOB              (wait for I/O register bit 11 to be set to 0, i.e. for the enable input to be inactive)
     WT IOB                (wait for I/O register bit 11 to be set to 1, i.e. for the enable input to be active)
     RSF                   (reset all faults)
     EX1                   (execute program 1)
     ED                    (end program 4 and exit program editor)
```

Once the IMC-3 model C is powered up, the controller will be faulted. The controller will automatically execute program 4, which waits until the enable input is inactive and then active. It then resets all controller faults and executes program 1, which waits for a positive edge sensitive digital input from 1 through 6 to be active. It will then execute the respective subroutine at 10, 20, 30, 40, 50 or 60, which will execute a specified motion.