YASKAWA

A World of Automation Solutions™

SMC–4000

User's Manual

Upon receipt of the product and prior to initial operation, read these instructions thoroughly, and retain for future reference

⚠ WARNING

YASKAWA manufactures component parts that can be used in a wide variety of industrial applications. The selection and application of YASKAWA products remain the responsibility of the equipment designer or end user. YASKAWA accepts no responsibility for the way its products may be incorporated into the final system design.

Under no circumstances should any YASKAWA product be incorporated into any product or design as the exclusive or sole safety control. Without exception, all controls should be designed to detect faults dynamically under all circumstances. All products designed to incorporate a component part manufactured by YASKAWA must be supplied to the end user with appropriate warnings and instructions as to that part's safe use and operation. Any warnings provided by Yaskawa must be promptly provided to the end user.

YASKAWA offers an express warranty only as to the quality of its products in conforming to standards and specifications published in YASKAWA'S manual. NO OTHER WARRANTY, EXPRESS OR IMPLIED, IS OFFERED. YASKAWA assumes no liability for any personal injury, property damage, losses or claims arising from misapplication of its products.

TABLE OF CONTENTS

1 Introduction

The SMC–4000 is a multi-axis Ethernet motion controller designed for use with Yaskawa's SIGMA series and LEGEND Digital Torque Amplifier.

It provides a structured text programming environment and the ability to perform many modes of motion including camming, gearing, and contouring. High speed product registration is also available as a standard feature.

Additionally, the Ethernet function allows multiple devices to communicate with the controller using a TELNET or MODBUS protocol.

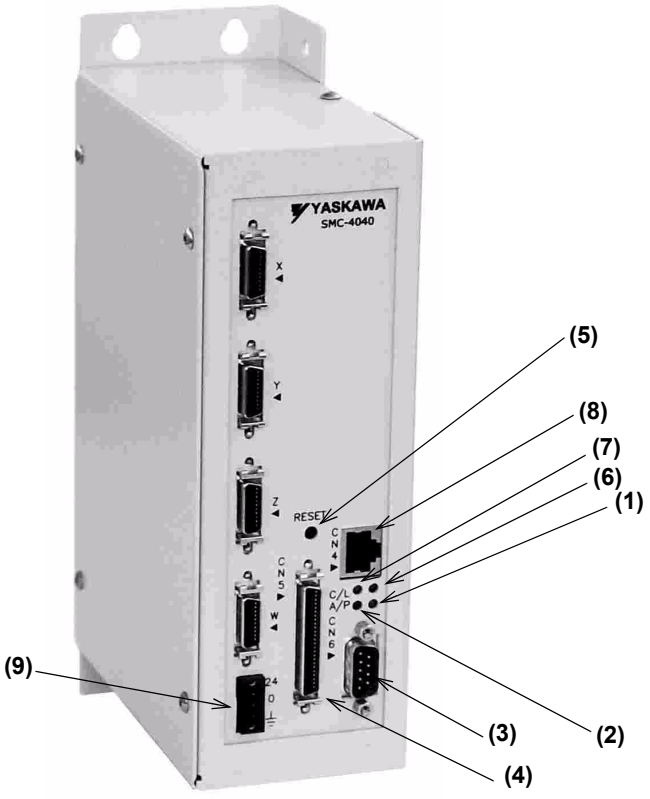
Part Numbers

		Description	Part Number
Controller	a)	Two axis motion controller	SMC4020
		Two axis motion controller w/ absolute encoder option	SMC4020W
		Four axis motion controller	SMC4040
		Four axis motion controller w/ absolute encoder option	SMC4040W
Encoder Cables	b)	Pigtail (5 feet)	SMCCBL005
		Pigtail (10 feet)	SMCCBL010
		Pigtail (15 feet)	SMCCBL015
		Prewired for SGD, SGDA, or SGDG (2 feet)	SMCCBL102
		Prewired for SGD, SGDA, or SGDG (5 feet)	SMCCBL105
		Prewired for SGD, SGDA, or SGDG (10 feet)	SMCCBL110
		Prewired for SGD, SGDA, or SGDG (15 feet)	SMCCBL115
		Prewired for SGDB or SGDH (2 feet)	SMCCBL202
		Prewired for SGDB or SGDH (5 feet)	SMCCBL205
		Prewired for SGDB or SGDH (10 feet)	SMCCBL210
		Prewired for SGDB or SGDH (15 feet)	SMCCBL215
		Prewired for SGDB or SGDH (2 feet) Includes Alarm & Reset	SMCCBLH02
		Prewired for SGDB or SGDH (5 feet) Includes Alarm & Reset	SMCCBLH05
		Prewired for SGDB or SGDH (10 feet) Includes Alarm & Reset	SMCCBLH10
		Prewired for SGDB or SGDH (15 feet) Includes Alarm & Reset	SMCCBLH15
		Prewired for SGD, SGDA, or SGDG (2 feet) with additional pigtail	SMCCBLA02
		Prewired for SGD, SGDA, or SGDG (5 feet) with additional pigtail	SMCCBLA05
		Prewired for SGD, SGDA, or SGDG (10 feet) with additional pigtail	SMCCBLA10
		Prewired for SGD, SGDA, or SGDG (15 feet) with additional pigtail	SMCCBLA15
		Prewired for SGDB or SGDH (2 feet) with additional pigtail	SMCCBLB02
		Prewired for SGDB or SGDH (5 feet) with additional pigtail	SMCCBLB05
		Prewired for SGDB or SGDH (10 feet) with additional pigtail	SMCCBLB10
		Prewired for SGDB or SGDH (15 feet) with additional pigtail	SMCCBLB15
I/O	c)	1.0m pigtail cable	JZSP-CKIO1-1(A)
		2.0m pigtail cable	JZSP-CKIO1-2(A)
		3.0m pigtail cable	JZSP-CKIO1-3(A)
		1.0m cable with OMRON terminal block	JUSP TA50P
		0.5m 50 pin I/O cable to DSUB	JZSP-CKIOD-D50
		1.0m 50 pin I/O cable to DSUB	JZSP-CKIOD-01
		2.0m 50 pin I/O cable to DSUB	JZSP-CKIOD-02
		CN5 Connector Kit (same as SGDH 1CN kit)	JZSP-CKI9
Serial	d)	2.0m CN6 serial port cable (included with YTerm software)	SMCCBL7
Software	e)	YTerm Integrated Development Environment	SMCGUI1
		SMCComm serial & ethernet driver for application development for all SMC products	SMCOCX1
Other	f)	Replacement power supply connector	UFS-0118

Start-up

Front Panel Description

No.	Name	Description
(1)	Power ON	A green LED that indicates power is being applied to the SMC–4000.
(2)	Alarm/Error	A red LED that will flash ON at power up and stay lit for approximately 2 seconds. After power up, the LED will illuminate for the following reasons: • An axis has a position error greater than the error limit. The error limit is set by using the ER command. • The reset input on the controller is held low or is being affected by noise. • There is a failure in the controller and the processor is resetting itself. • There is a failure in the output IC which drives the error signal.
(3)	CN6	9 pin male D-Sub serial port connector
(4)	CN5	3M 50 pin high density I/O connector
(5)	RST	Reset button. Causes the controller to reboot, and load the application program and parameters from flash. If the program contains an #AUTO label, it will automatically execute.
(6)	Ethernet status	A green LED that is lit when there is an Ethernet connection to the controller. This LED indicates physical connection, not active communication.
(7)	Ethernet status	The yellow LED indicates traffic across the Ethernet connection. This LED will show both transmit and receive activity across the connection. If there is no Ethernet connection or IP address assigned, the LED will flash at regular intervals to show that the BOOTP packets are being broadcast.
(8)	CN4	10 BaseT Ethernet RJ45 Connector
(9)		Power Connector (+24VDC, 0VDC, FG⏚



Axis Connector (20–pin)

SMC Axis Connector			SGDH CN1
PIN	SIGNAL	Reference	Pin
1	PA	input	33
2	/PA	input	34
3	PB	input	35
4	/PB	input	36
5	PC	input	19
6	/PC	input	20
7	Motor Command	output	9
8	+5 / +12 / -12 Common	output	10
9	+5 / +12 / -12 Common	output	2
10	+5 / +12 / -12 Common	output	6
11	Amplifier Enable	output	40
12	Step	output	11
13	Sen/Dir	output	4
14	+5 / +12 / -12 Common	output	1
15	Alarm +	input	31
16	Reset	output	44
17	ALM -	input	32
18	n/c		
19	+24 VDC	output	47
20	n/c		

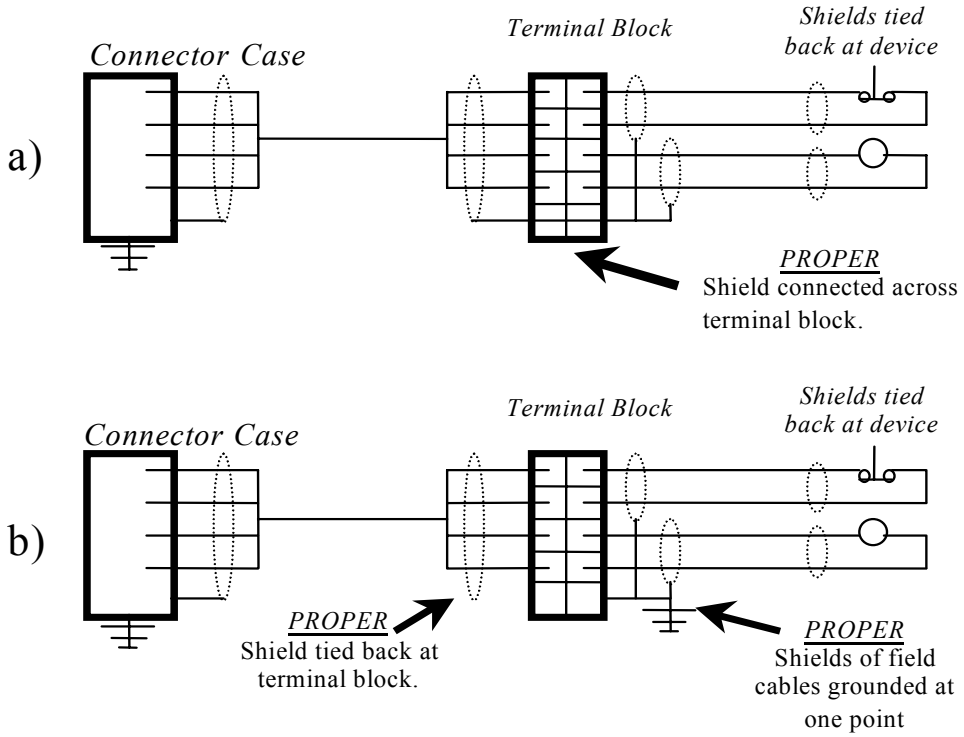
I/O Connections (50-pin CN5)

SMC Output Connector CN5	
PIN (Numerical Order)	SIGNAL
1	Home W
2	Home Z
3	Home Y
4	Home X
5	Input 1
6	Input 4
7	Input 7
8	Output 3
9	Output 5
10	Output 8
11	X Aux Encoder A+
12	X Aux Encoder B-
13	Y Aux Encoder B+
14	Reverse Limit W
15	Reverse Limit Z
16	Reverse Limit Y
17	Reverse Limit X
18	Input 2
19	Input 5
20	Input 8
21	Output 2
22	Output 7
23	X Aux Encoder A-
24	Y Aux Encoder A+
25	Y Aux Encoder B-
26	Reset
27	Forward Limit W
28	Forward Limit Z
29	Forward Limit Y
30	Forward Limit X
31	Input 3
32	Input 6
33	Abort
34	Output 1
35	Output 4
36	Output 6
37	X Aux Encoder B+
38	Y Aux Encoder A-
39	E-Stop1
40	E-Stop2
41	Z Aux Encoder A+
42	Z Aux Encoder A-
43	Z Aux Encoder B+
44	Z Aux Encoder B-
45	W Aux Encoder A+
46	W Aux Encoder A-
47	W Aux Encoder B+
48	W Aux Encoder B-
49	Spare 1
50	Spare 2

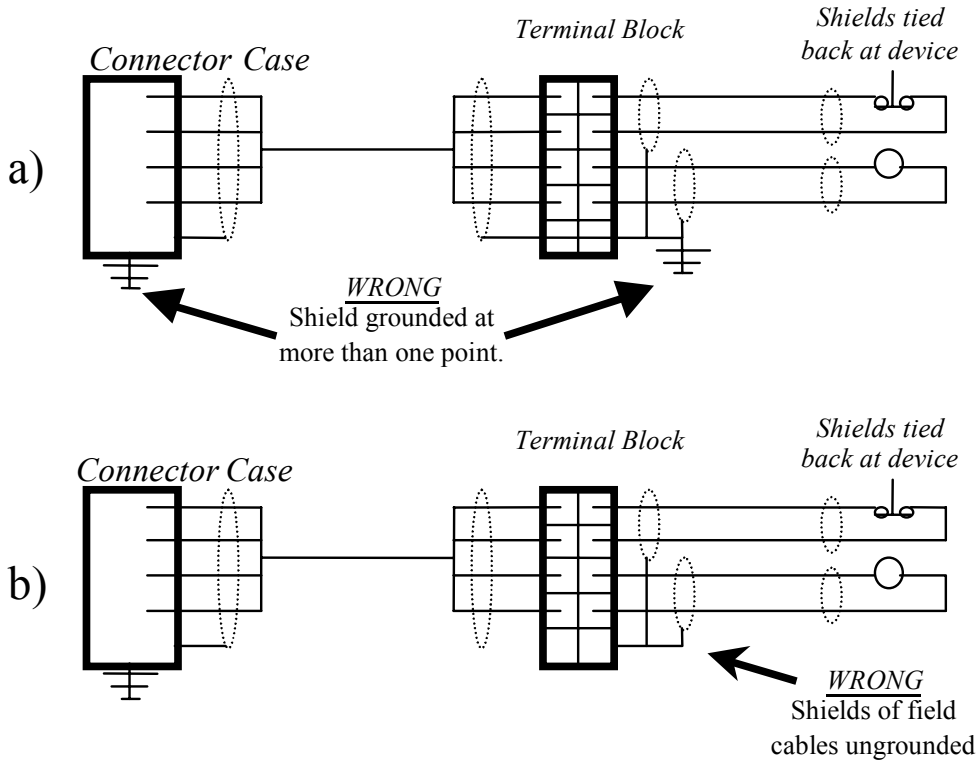
SMC Output Connector CN5	
SIGNAL (Alphabetical Order)	PIN
Abort	33
E-Stop1	39
E-Stop2	40
Forward Limit W	27
Forward Limit X	30
Forward Limit Y	29
Forward Limit Z	28
Home W	1
Home X	4
Home Y	3
Home Z	2
Input 1	5
Input 2	18
Input 3	31
Input 4	6
Input 5	19
Input 6	32
Input 7	7
Input 8	20
Output 1	34
Output 2	21
Output 3	8
Output 4	35
Output 5	9
Output 6	36
Output 7	22
Output 8	10
Reset	26
Reverse Limit W	14
Reverse Limit X	17
Reverse Limit Y	16
Reverse Limit Z	15
Spare 1	49
Spare 2	50
W Aux Encoder A-	46
W Aux Encoder A+	45
W Aux Encoder B-	48
W Aux Encoder B+	47
X Aux Encoder A-	23
X Aux Encoder A+	11
X Aux Encoder B-	12
X Aux Encoder B+	37
Y Aux Encoder A-	38
Y Aux Encoder A+	24
Y Aux Encoder B-	25
Y Aux Encoder B+	13
Z Aux Encoder A-	42
Z Aux Encoder A+	41
Z Aux Encoder B-	44
Z Aux Encoder B+	43

Cable Shielding, Segregation and Noise Immunity

Proper



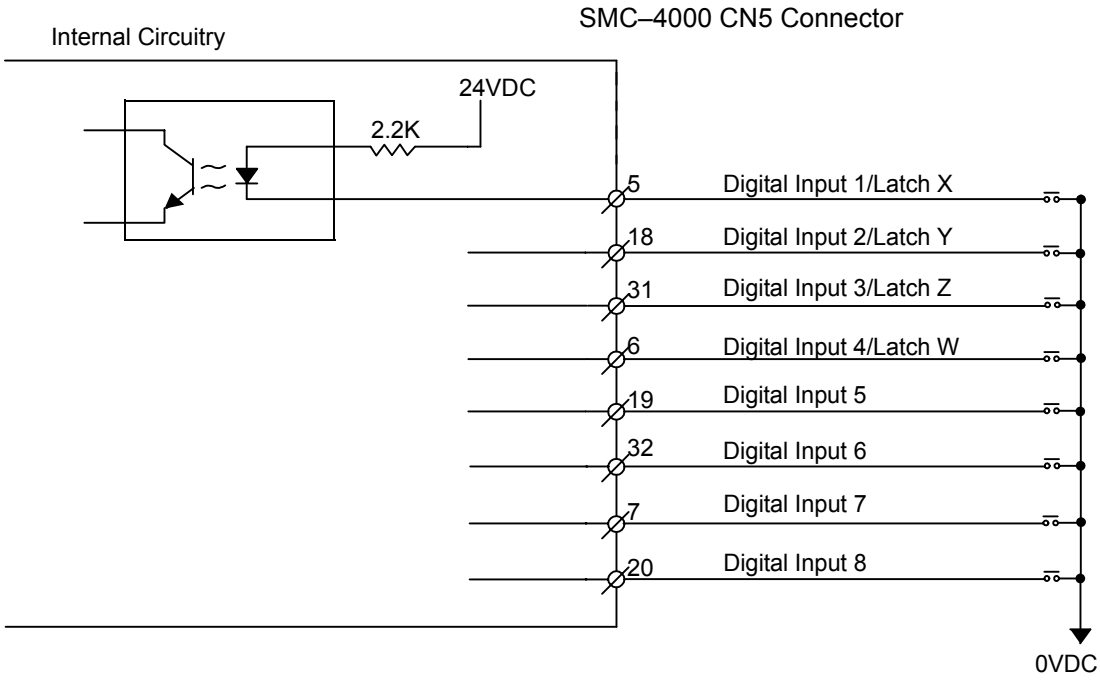
Wrong



Digital I/O

Digital Input

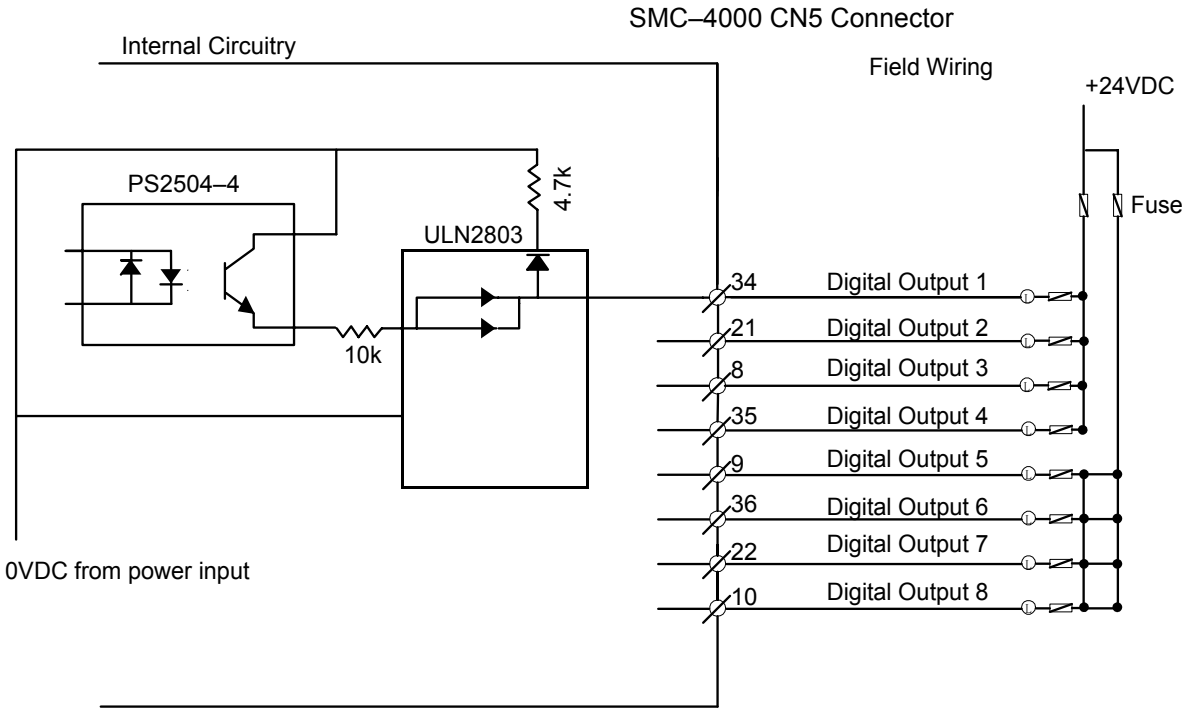
Item	Specifications
Number of Input Points	8
Input Format	Sinking
Isolation	Optical
Voltage	24VDC ± 20%
Current Rating (ON)	5.3mA to activate
Input Impedance	2.2kΩ
Operation Voltage	Logic 0 <5V Logic 1 >15V
OFF Current	0.9mA or less
Response Time (Hardware)	OFF to ON: <0.5ms ON to OFF: <1.5ms
Latch response time	Less than 25µsec
Minimum latch width	9µsec
NOTE: Inputs float high unless the input is held low.	



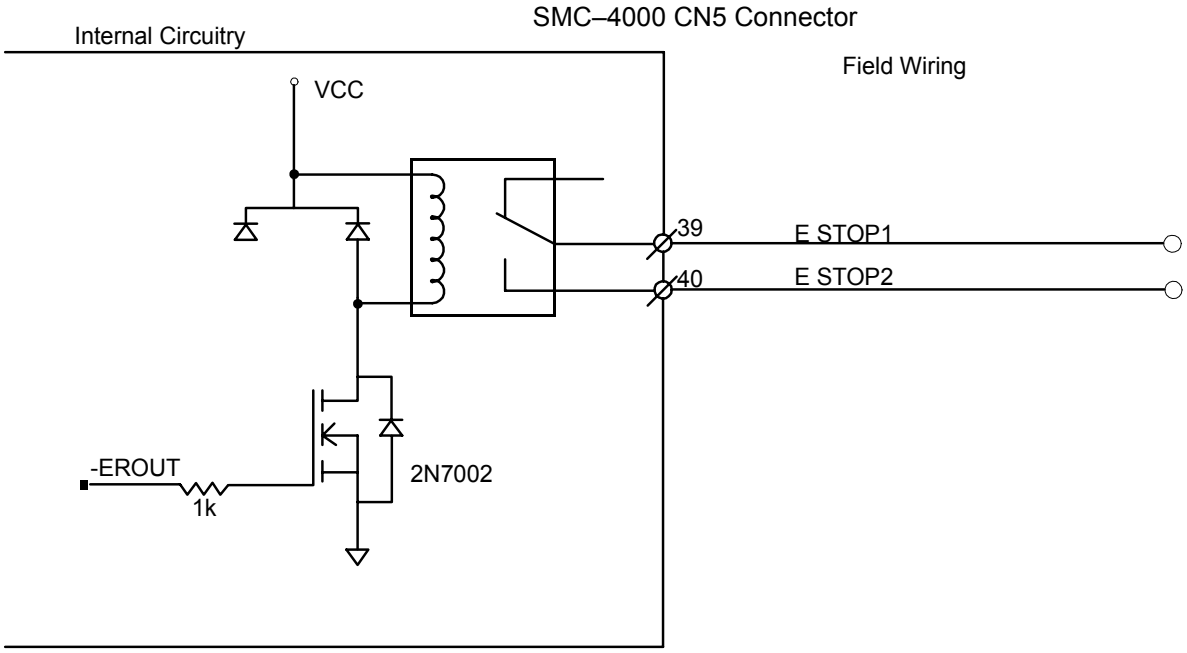
Digital Output

Item	Specifications
Number of Output Points	8
Output Format	Sinking
Output Classification	Transistor Output
Isolation	Optical
Load Voltage	24VDC ± 20%
Load Current	200mA/Output (600mA if activated individually)
Response Time	OFF to ON <0.25ms ON to OFF <0.5ms
External Common Power	24VDC ± 20% 15mA
Common User Fuse Rating	800mA per bank of four
Individual User Fuse Rating	200mA recommended

NOTE: The ULN 2803 output chip is capable of 600mA at a single output, or 800mA for the eight outputs simultaneously.



Emergency Stop Chain



The SMC–4000 closes the relay contact under normal operating conditions. The relay is controlled by the same circuit as the error LED. The relay will be open if the error LED is ON.

Ratings:

1.0A @ 24VDC

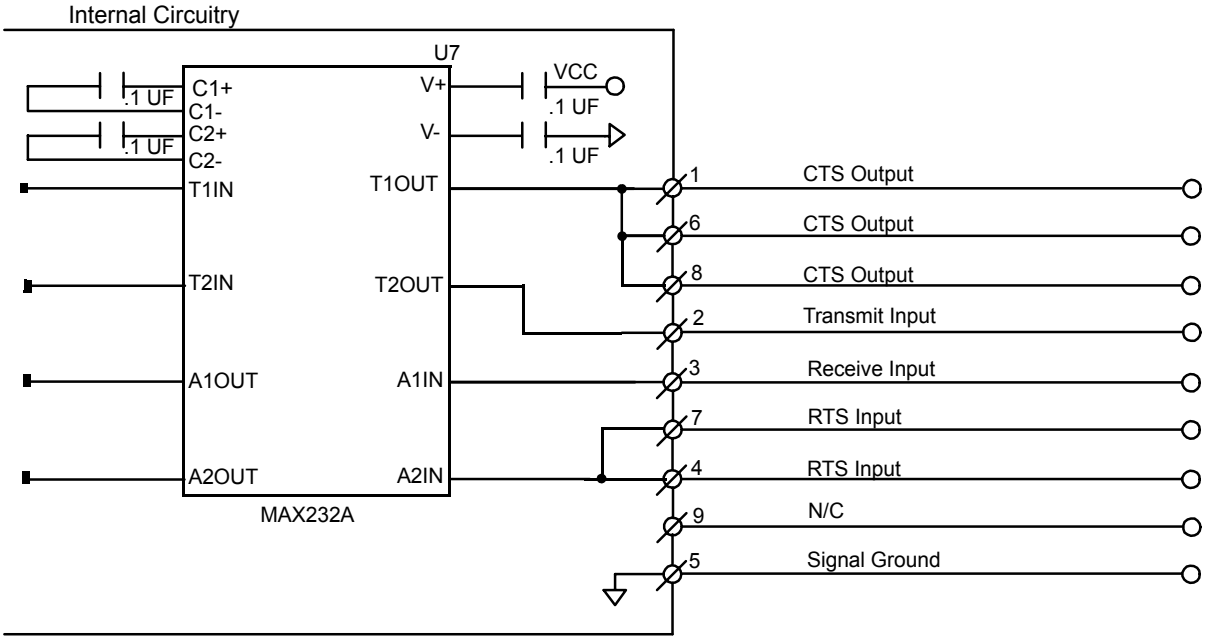
0.5A @ 125VDC

Maximum switching power: 62.5VA, 30W

Serial Communication

Item	Specifications
Baud Rate	9600 or 19200 settable by jumper JP2, default is 19200
Data Bits	8
Parity	None
Stop Bits	1

SMC–4000 Serial Port 6CN

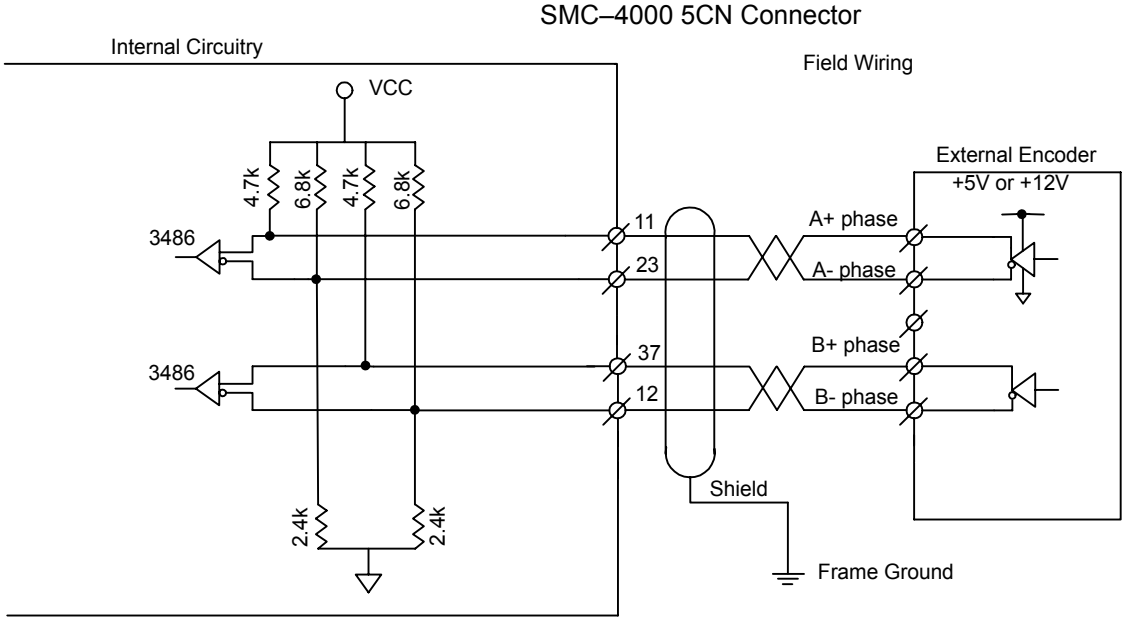


NOTE: Hardware handshaking must be used with the SMC–4000. If it is impossible to implement hardware handshaking, use a jumper between pins 1 and 4 in the connector.

NOTE: Do not connect pin 5 to a 24V ground. This would defeat the opto isolation.

External Encoder Specifications

Item	Specifications
Number of External Encoders	One per Main Axis
Input Format	Quadrature or Pulse and Direction
Maximum Frequency	12 MHz
Current Draw	940 µAmp

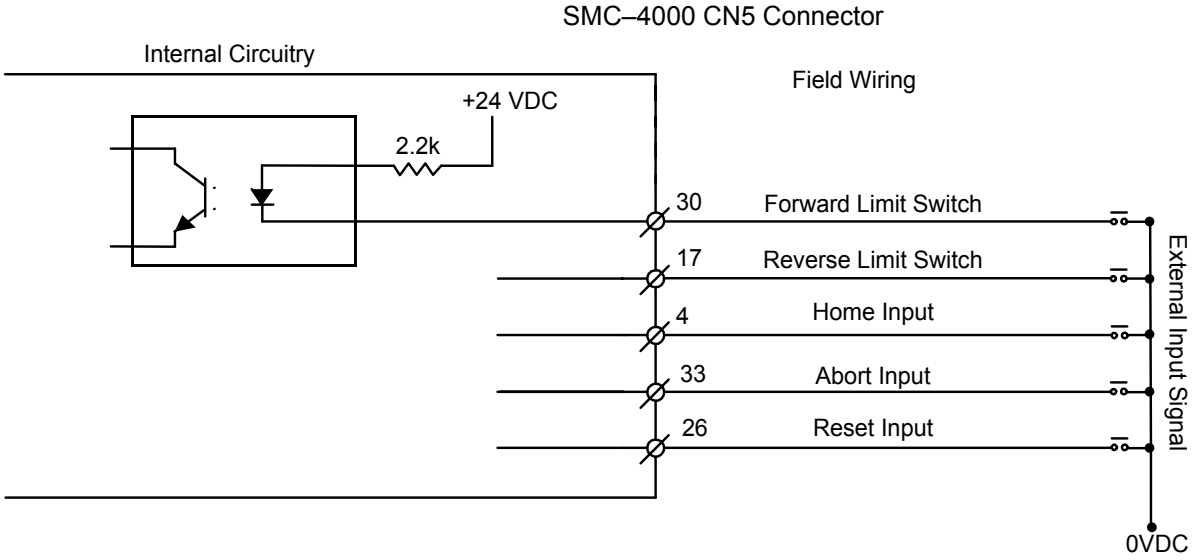


X axis internal encoder shown
See 5CN connector for other axis pin connections.

Standard voltage levels are TTL (0V to 5V), however, voltage levels up to 12V are acceptable. If using differential 12V signals, no modification is required. Single ended 12V signals require a bias voltage applied to the complimentary input, i.e.; use two 10k resistors, one connected to +12V and the other connected to the encoder signal ground to hold the /A phase and /B phase at 6VDC. Do not use a 24VDC encoder.

Dedicated Inputs

Item	Specifications
Number of Input Points	Forward limit, Reverse limit, Home for all axes; and Abort, Reset
Input Format	Sinking
Isolation	Optical
Voltage	24 VDC ± 20%
Current Rating (ON)	5.3 mA to activate
Input Impedance	2.2k Ω
Operation Voltage	Logic 0 <5V Logic 1 >15V
OFF Current	0.9 mA or less
Limit Switch Response Time	OFF to ON: <0.5 ms ON to OFF: <1.5 ms



X axis dedicated inputs shown. Other axes are the same.
Check CN5 connector for pin configuration.

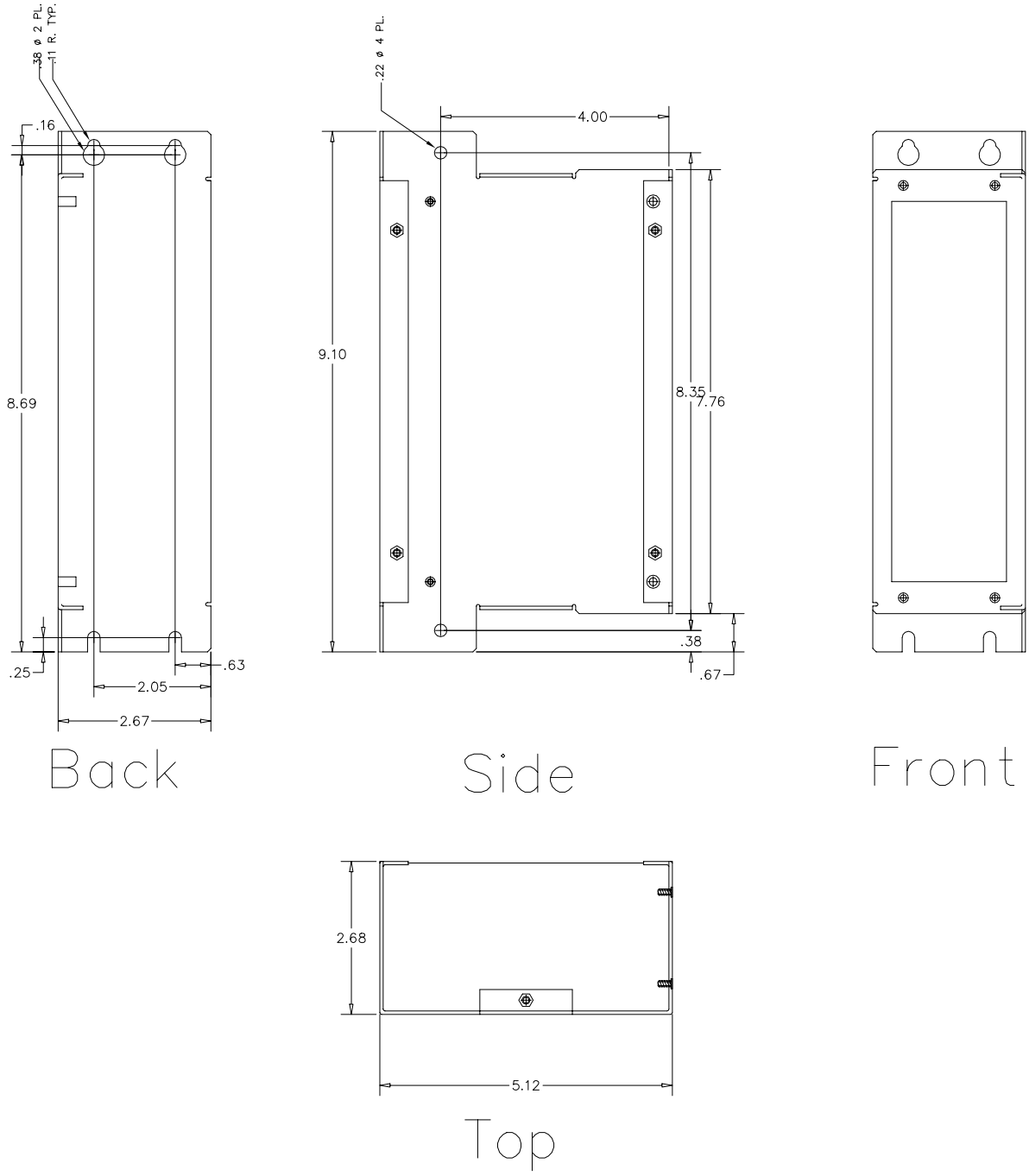
Physical Specifications

Description	Specifications
Depth	5 inches
Width	2.6 inches
Height	9.1 inches
Weight	3.52lbs (1.6kg)
Vibration	9.8 $msec^2$ (1.0g)
Ambient Temperature	0 ~ 70° C (32 ~ 158° F)
Humidity	Less than 95%
Noise	IEC Level 3

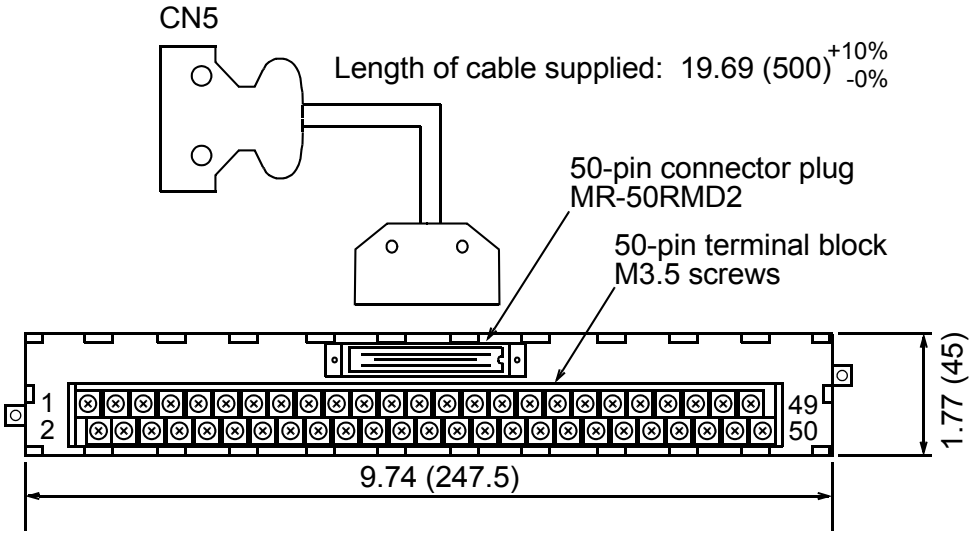
Hardware Specifications

Description	Specifications
CPU	25MHz Motorola
Servo Update	1000µs default, 250µs minimum
Digital Inputs	(8), +24VDC
Dedicated Inputs	(2) +24VDC +3 per axis @24VDC
Digital Outputs	(8), +24VDC
Serial Port	(1) 9600 or 19200 baud
Ethernet	(1) 10-base-T
Power Inpu	24 VDC – 600mA

Dimensional Drawings

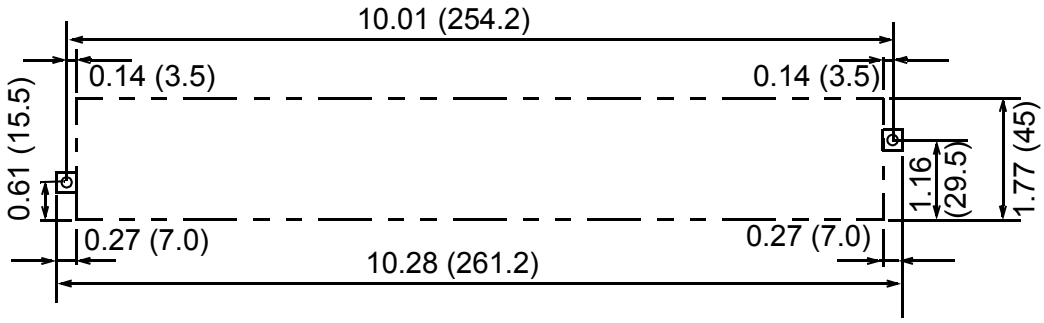


I/O Cable with Terminal Block JUSP-TA50P



Connector Terminal Block Converter Unit
JUSP-TA50P* (cable included)

Mounting Hole Diagram



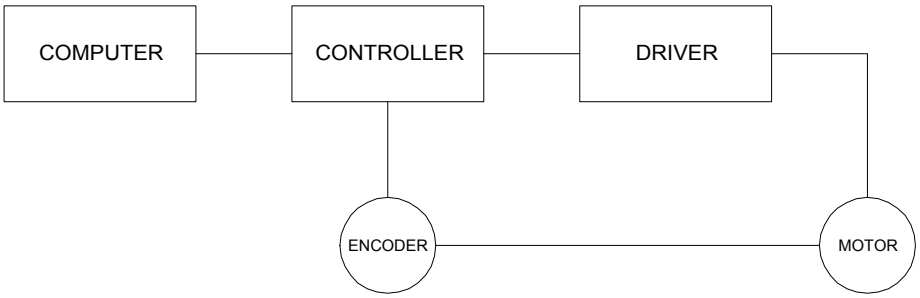
*Terminal specifications: See “I/O Connections (50-pin CN5)” on page 5.

NOTES

2 Theory of Operation

Overview

The following discussion covers the operation of motion control systems. A typical motion control system consists of the elements shown in the following illustration:



Elements of Servo Systems

The operation of such a system can be divided into three levels, as shown in the following illustration *Levels of Control Functions*. The levels are:

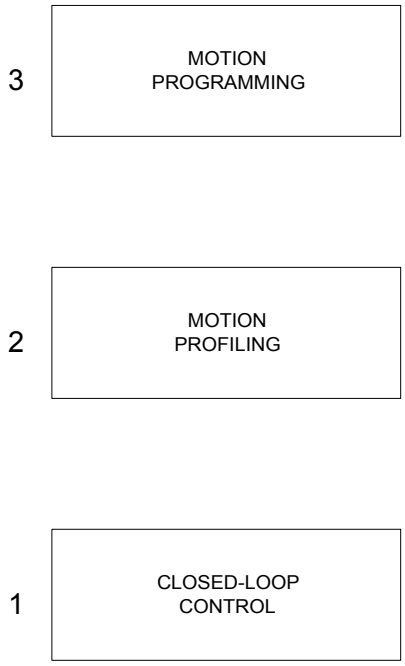
1. Closing the Loop
2. Motion Profiling
3. Motion Programming

The first level, the closing of the loop, assures that the motor follows the commanded position. Closing the position loop using a sensor does this. The operation at the basic level of closing the loop involves the subjects of modeling, analysis, and design. These subjects will be covered in the following discussions.

The motion profiling is the generation of the desired position function. This function, R(t), describes where the motor should be at every sampling period. Note that the profiling and the closing of the loop are independent functions. The profiling function determines where the motor should be and the closing of the loop forces the motor to follow the commanded position

The highest level of control is the motion program. This can be stored in the host computer or in the controller. This program describes the tasks in terms of the motors that need to be controlled, the distances and the speed.

Levels



Levels of Control Functions

The three levels of control may be viewed as different levels of management. The top manager, the motion program, may specify the following instruction, for example.

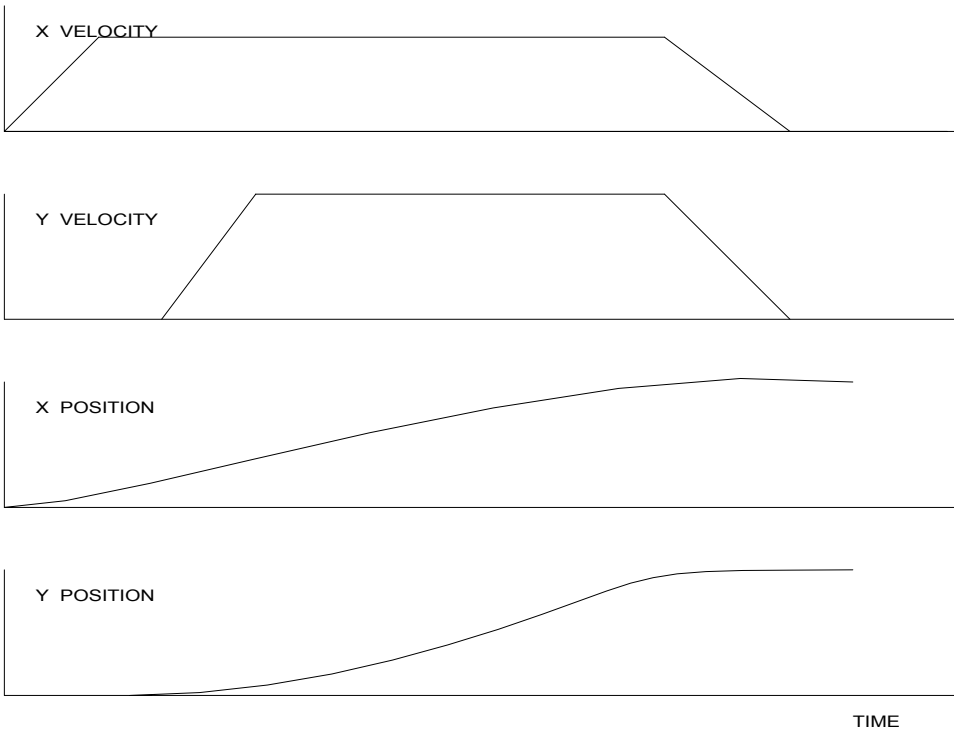
PR 6000,4000
SP 20000,20000
AC 200000,300000
BG X
AD 2000
BG Y
EN

This program corresponds to the velocity profiles shown in the following illustration - *Velocity and Position Profiles*. Note that the profiled positions show where the motors must be at any instant of time.

Finally, it remains up to the servo system to verify that the motor follows the profiled position by closing the servo loop.

The operation of the servo system is done in two manners. First, it is explained qualitatively, in the following section. Later, the explanation is repeated using analytical tools for those who are more theoretically inclined.

Velocity and Position Profiles



Operation of Closed-Loop Systems

To understand the operation of a servo system, we may compare it to a familiar closed-loop operation, adjusting the water temperature in the shower. One control objective is to keep the temperature at a comfortable level, say 90 degrees F. To achieve that, our skin serves as a temperature sensor and reports to the brain (controller). The brain compares the actual temperature, which is called the feedback signal, with the desired level of 90 degrees F. The difference between the two levels is called the error signal. If the feedback temperature is too low, the error is positive, and it triggers an action which raises the water temperature until the temperature error is reduced sufficiently.

The closing of the servo loop is very similar. Suppose that we want the motor position to be at 90 degrees. A position sensor, often an encoder, measures the motor position and the position feedback is sent to the controller. Like the brain, the controller determines the position error, which is the difference between the commanded position of 90 degrees and the position feedback. The controller then outputs a signal that is proportional to the position error. This signal produces a proportional current in the motor, which causes a motion until the error is reduced. Once the error becomes small, the resulting current will be too small to overcome the friction, causing the motor to stop.

The analogy between adjusting the water temperature and closing the position loop carries further. We have all learned that the hot water faucet should be turned at the "right" rate. If you turn it too slowly, the temperature response will be slow, causing discomfort. Such a slow reaction is called overdamped response.

The results may be worse if we turn the faucet too fast. The overreaction results in temperature oscillations. When the response of the system oscillates, we say that the system is unstable. Clearly, unstable responses are bad when we want a constant level.

What causes the oscillations? The basic cause for the instability is a combination of delayed reaction and high gain. In the case of the temperature control, the delay is due to the water flowing in the pipes. When the human reaction is too strong, the response becomes unstable.

Servo systems also become unstable if their gain is too high. The delay in servo systems is between the application of the current and its effect on the position. Note that the current must be applied long enough to cause a significant effect on the velocity, and the velocity change must last long enough to cause a position change. This delay, when coupled with high gain, causes instability.

This motion controller includes a special filter that is designed to help the stability and accuracy. Typically, such a filter produces, in addition to the proportional gain, damping and integrator. The combination of the three functions is referred to as a PID filter.

The filter parameters are represented by the three constants KP, KI and KD, which correspond to the proportional, integral and derivative term respectively.

The damping element of the filter acts as a predictor, thereby reducing the delay associated with the motor response.

The integrator function, represented by the parameter KI, improves the system accuracy. With the KI parameter, the motor does not stop until it reaches the desired position exactly, regardless of the level of friction or opposing torque.

The integrator also reduces the system stability. Therefore, it can be used only when the loop is stable and has a high gain.

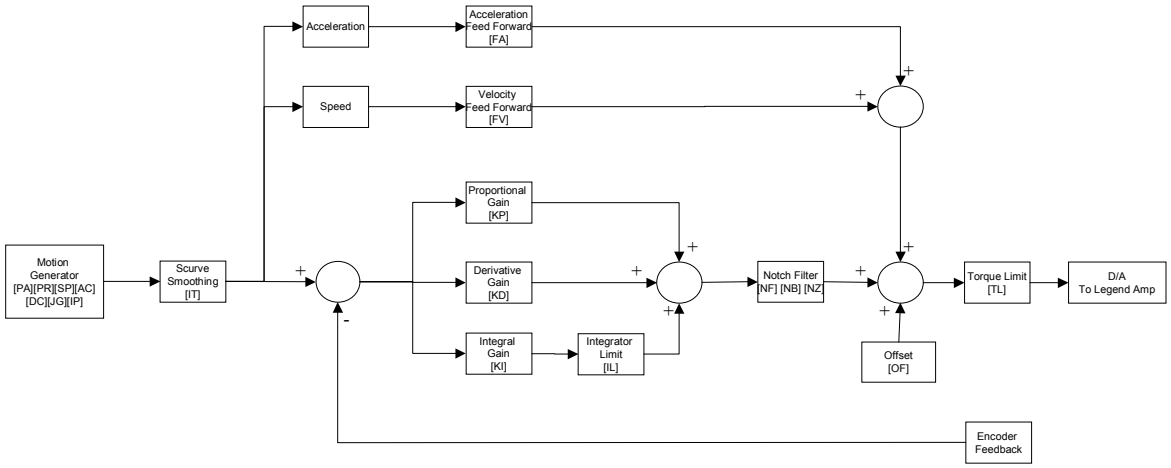
The output of the filter is applied to a digital-to-analog converter (DAC). The resulting output signal in the range between +10 and -10 Volts is then applied to the amplifier and the motor.

The motor position, whether rotary or linear is measured by a sensor. The resulting signal, called position feedback, is returned to the controller for closing the loop.

The following section describes the operation in a detailed mathematical form, including modeling, analysis and design.

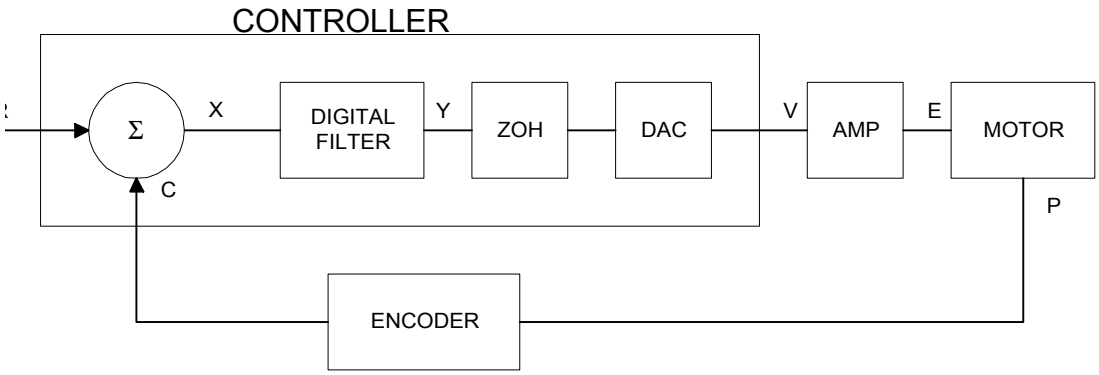
System Modeling

Basic Block Diagram



The elements of a servo system include the motor, driver, encoder and the controller. These elements are shown in the following illustration. The mathematical model of the various components is given below:

Controller



Functional Elements of a Motion Control System

Motor-Amplifier

The motor amplifier is configured for current mode:

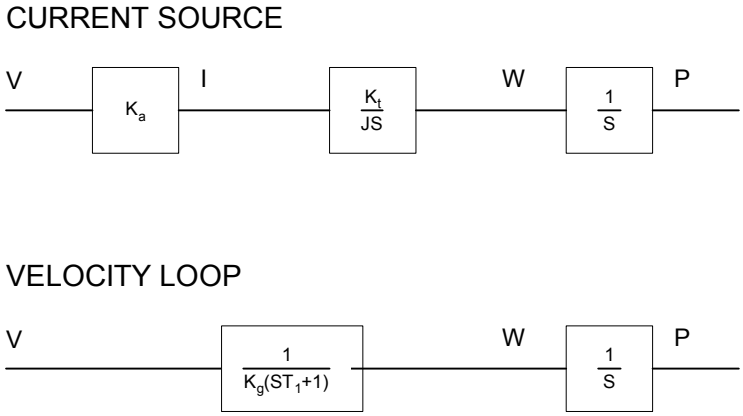
Current Drive

The current drive generates a current I, which is proportional to the input voltage, V, with a gain of Ka, a torque constant of K_t, and inertia J. The resulting transfer function in this case is:

$$P/V = K_a K_t / Js^2$$

For example, a current amplifier with K_a = 2 A/V with the motor described by the previous example will have the transfer function:

$$P/V = 1000/s^2 \qquad [rad/V]$$



Mathematical model of the motor and amplifier in two operational modes

Encoder

The encoder generates N pulses per revolution. It outputs two signals, Channel A and B, which are in quadrature. Due to the quadrature relationship between the encoder channels, the position resolution is increased to 4N quadrature counts/rev.

The model of the encoder can be represented by a gain of:

$$K_f = 4N/2\pi \qquad [count/rad]$$

For example, a 1000 lines/rev encoder is modeled as:

$$K_f = 638$$

DAC

The DAC or D-to-A converter converts a 16-bit number to an analog voltage. The input range of the numbers is 65538and the output voltage range is +/-10V or 20V. Therefore, the effective gain of the DAC is:

$$K= 20/65538 = 0.000305 \qquad [V/count]$$

Digital Filter

The digital filter has a transfer function of D(z) = K(z-A)/z + Cz/z-1 and a sampling time of T.

The filter parameters, K, A and C are selected by the instructions KP, KD, KI or by GN, ZR and KI, respectively. The relationship between the filter coefficients and the instructions are:

K = KP + KD	or K = GN
A = KD/(KP + KD)	or A = ZR
C = KI/8	

This filter includes a lead compensation and an integrator. It is equivalent to a continuous PID filter with a transfer function G(s).

$$G(s) = P + sD + I/s$$

$$P = K(1-A) = KP$$

$$D = T* K * A = T.KD$$

$$I = C/T = KI/8 * TM$$

For example, if the filter parameters are KP = 4:

KD = 36

KI = 2

T = 0.001 s

the digital filter coefficients are:

K = 40

A = 0.9

C = 0.25

and the equivalent continuous filter, G(s), is:

$$G(s) = 4 + 0.036s + 250/s$$

ZOH

The ZOH, or zero-order-hold, represents the effect of the sampling process, where the motor command is updated once per sampling period. The effect of the ZOH can be modeled by the transfer function

$$H(s) = 1/(1+sT/2)$$

If the sampling period is T = 0.001, for example, H(s) becomes:

$$H(s) = 2000/(s+2000)$$

However, in most applications, H(s) may be approximated as one.

This completes the modeling of the system elements. Next, we discuss the system analysis.

System Analysis

To analyze the system, we start with a block diagram model of the system elements. The analysis procedure is illustrated in terms of the following example.

Consider a position control system with the SMC–4000 controller and the following parameters:

$K_t = 0.1$	Nm/A	Torque constant
$J = 2.10^{-4}$	$kg.m^2$	System moment of inertia
R = 2	W	Motor resistance
$K_a = 4$	Amp/Volt	Current amplifier gain
KP = 12.5		Digital filter gain
KD = 245		Digital filter zero
KI = 0		No integrator
N = 500	Counts/rev	Encoder line density
T = 1	ms	Sample period

The transfer function of the system elements are:

Motor:

$$M(s) = P/I = Kt/Js2 = 500/s^2 \ \ [rad/A]$$

Amp:

$$K_a = 4 \ \ [Amp/V]$$

DAC:

$$K_d = 0.0012 \ \ [V/count]$$

Encoder:

$$K_f = 4N/2\pi = 318 \ [count/rad]$$

ZOH:

$$2000/(s+2000)$$

Digital Filter:

$$KP = 12.5, \ KD = 245, \ T = 0.001$$

Therefore,:

$$D(z) = 12.5 + 245 \ (1-z-1)$$

Accordingly, the coefficients of the continuous filter are:

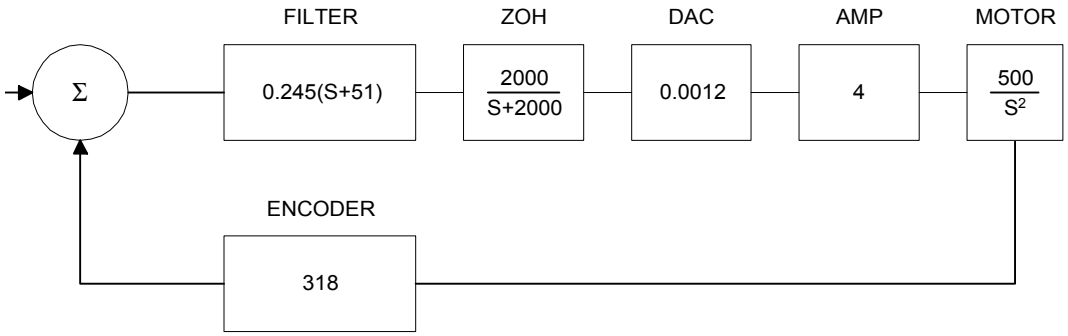
$$P = 12.5$$

$$D = 0.245$$

The filter equation may be written in the continuous equivalent form:

$$G(s) = 12.5 + 0.245s = 0.245(s+51)$$

The system elements are shown in the following illustration:

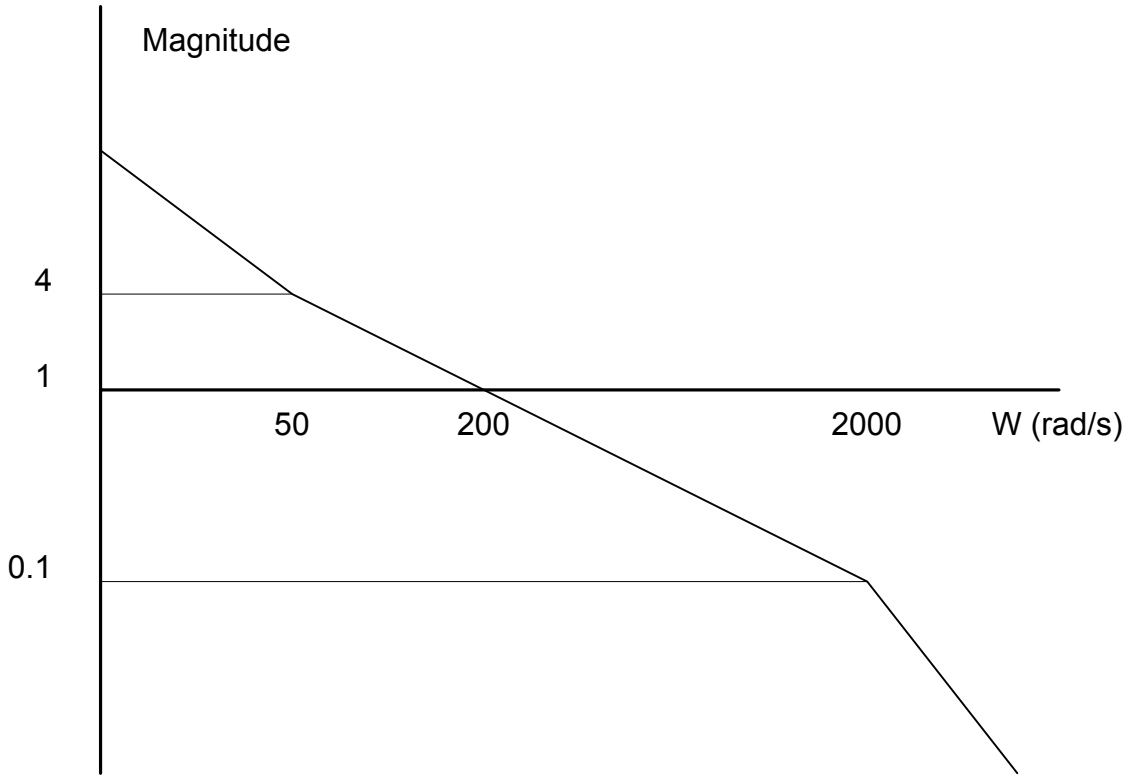


Mathematical model of the control system

The open loop transfer function, A(s), is the product of all the elements in the loop:

$$A = 390{,}000\ (s+51)/[s^2(s+2000)]$$

To analyze the system stability, determine the crossover frequency, ω_c at which A(j ω_c) equals one. This can be done by the Bode plot of A(j ω_c), as shown in the following illustration:



Bode plot of the open loop transfer function

For the given example, the crossover frequency was computed numerically resulting in 200 rad/s.

Next, we determine the phase of A(s) at the crossover frequency:

$$A(j200) = 390{,}000\ (j200+51)/[(j200)^2 * (j200 + 2000)]$$

$$\alpha = \text{Arg}[A(j200)] = \tan^{-1}(200/51) - 180° - \tan^{-1}(200/2000)$$

$$\alpha = 76° - 180° - 6° = -110°$$

Finally, the phase margin, PM, equals:

$$PM = 180° + \alpha = 70°$$

As long as PM is positive, the system is stable. However, for a well damped system, PM should be between 30 degrees and 45 degrees. The phase margin of 70 degrees given above indicated overdamped response.

Next, we discuss the design of control systems.

System Design and Compensation

The closed-loop control system can be stabilized by a digital filter, which is pre-programmed in the SMC–4000 controller. The filter parameters can be selected by the user for the best compensation. The following discussion presents an analytical design method.

The Analytical Method

The analytical design method is aimed at closing the loop at a crossover frequency, ω_c, with a phase margin PM. The system parameters are assumed known. The design procedure is illustrated by a design example.

Consider a system with the following parameters:

K_t	Nm/A	Torque constant
$J = 2 * 10^{-4}$	$kg.m^2$	System moment of inertia
R = 2	W	Motor resistance
$K_a = 2$	Amp/Volt	Current amplifier gain
N = 1000	Counts/rev	Encoder line density

The DAC of the SMC–4000 outputs +/-10V for a 16-bit command of +/-32768 counts.

The design objective is to select the filter parameters in order to close a position loop with a crossover frequency of ω_c = 500 rad/s and a phase margin of 45 degrees.

The first step is to develop a mathematical model of the system, as discussed in the previous system.

Motor:

$$M(s) = P/I = K_t/Js^2 = 1000/s^2$$

Amp:

$$K_a = 2 \qquad [Amp/V]$$

DAC

$$K_d = 10/8192:$$

Encoder:

$$K_f = 4N/2\pi = 636$$

ZOH:

$$H(s) = 2000/(s+2000)$$

Compensation Filter:

$$G(s) = P + sD$$

The next step is to combine all the system elements, with the exception of G(s), into one function, L(s):

$$SL(s) = M(s)\, K_a\, K_d\, K_f\, H(s) = 1.27*10^7/[s^2(s+2000)]$$

Then the open loop transfer function, A(s), is:

$$A(s) = L(s)\, G(s)$$

Now, determine the magnitude and phase of L(s) at the frequency $\omega_c = 500$:

$$L(j500) = 1.27*10^7/[(j500)^2 (j500+2000)]$$

This function has a magnitude of:

$$|L(j500)| = 0.025$$

and a phase:

$$Arg[L(j500)] = -180° - \tan^{-1}(500/2000) = -194°$$

G(s) is selected so that A(s) has a crossover frequency of 500 rad/s and a phase margin of 45 degrees. This requires that:

$$|A(j500)| = 1$$

$$Arg\ [A(j500)] = -135°$$

However, since:

$$A(s) = L(s)\ G(s)$$

then it follows that G(s) must have magnitude of:

$$|G(j500)| = |A(j500)/L(j500)| = 40$$

and a phase:

$$arg\ [G(j500)] = arg\ [A(j500)] - arg\ [L(j500)] = -135° + 194° = 59°$$

In other words, we need to select a filter function G(s) of the form:

$$G(s) = P + sD$$

so that at the frequency $\omega_c = 500$, the function would have a magnitude of 40 and a phase lead of 59 degrees.

These requirements may be expressed as:

$$|G(j500)| = |P + (j500D)| = 40$$

and:

$$arg\ [G(j500)] = \tan^{-1}[500D/P] = 59°$$

The solution of these equations leads to:

$$P = 40\cos 59° = 20.6$$

$$500D = 40\sin 59° = 34.3$$

Therefore:

$$D = 0.0686$$

and:

$$G = 20.6 + 0.0686s$$

The function G is equivalent to a digital filter of the form:

$$D(z) = KP + KD(1-z^{-1})$$

where:

$$KP = P$$

and:

$$KD = D/T$$

Assuming a sampling period of T=1ms, the parameters of the digital filter are:

KPX = 20.6

KDX = 68.6

The SMC–4000 can be programmed with the instruction:

KP 20.6

KD 68.6

In a similar manner, other filters can be programmed. The procedure is simplified by the following table, which summarizes the relationship between the various filters.

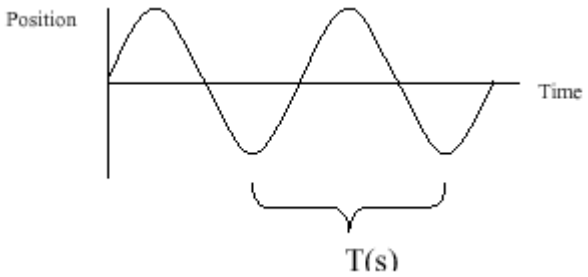
Digital	D(z) = K(z-A/z) + Cz/z-1
Digital	$D(z) = KP + KD(1-z^{-1}) + KI/8(1-z^{-1})$
KP, KD, KI	K = KP + KD A = KD/(KP+KD) C = KI/8
Digital	D(z) = GN(z-ZR)/z + KI z/8(z-1)
GN, ZR, KI	K = GN A = ZR C = KI/8
Continuous	G(s) = P + Ds + I/s
PID, T	P = K(1-A) = KP D = K *A * T = T * KD I = C/T = KI / 8 * TM

Notch Filter

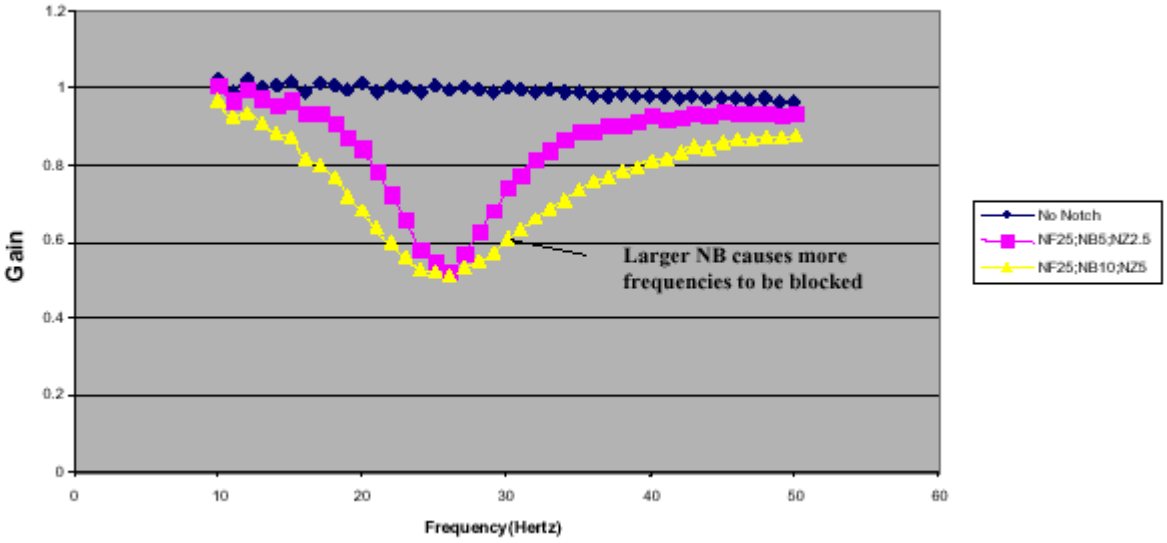
There are some applications in which the standard tuning procedure using the PID filter of the controller cannot completely eliminate the resonance in a system. Resonance occurs when the natural frequency of a system is excited in a way that increases the amplitude of oscillation. This is usually due to system compliance, such as a mechanical coupling or inherent motor characteristics.

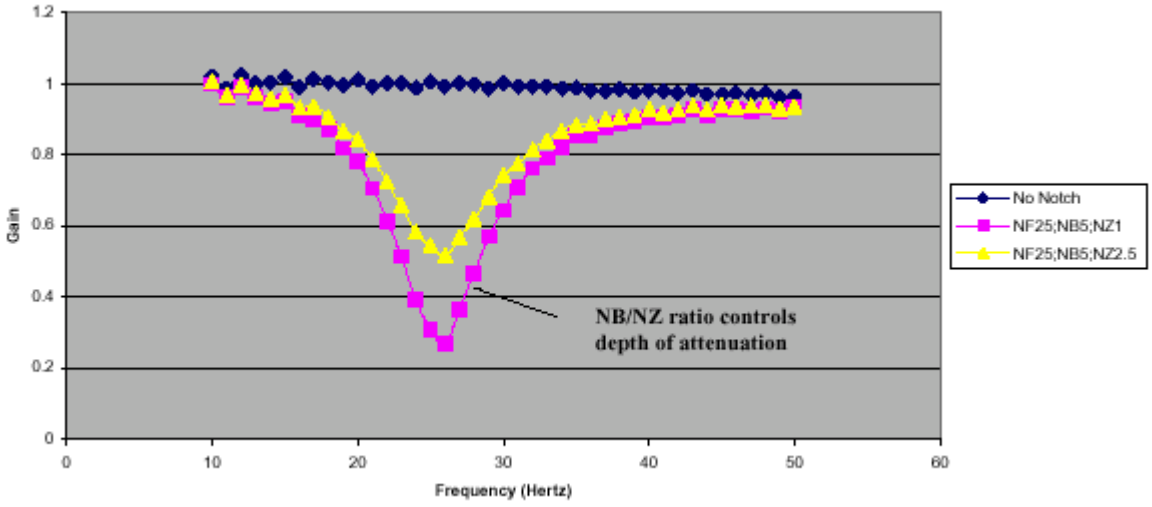
The notch filter is an advanced tuning technique that acts much like a “band-reject” filter in an electronic circuit. Certain frequencies are rejected while others are allowed to pass through. This is particularly helpful when trying to eliminate a resonance that always occurs at a single frequency.

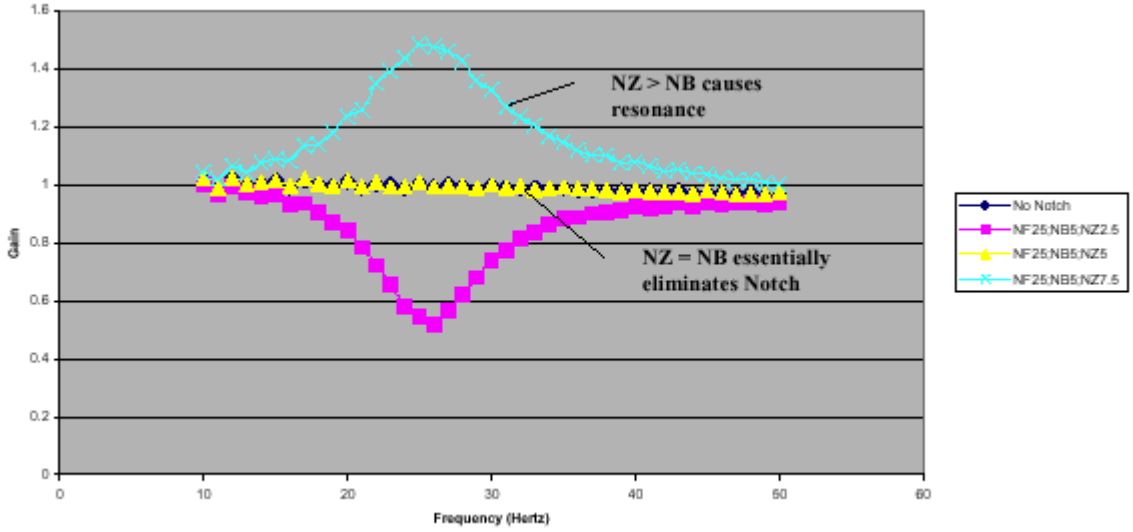
If a system oscillates at a specific point, then the first thing to do is find out at what frequency it occurs. The easiest way to do this is to graph the Actual Motor Position versus Time while the motor is oscillating. A sine wave with a constant frequency of oscillation should be seen. To get the frequency, f (Hertz), count the number of peaks that occur in 1 second. Or alternatively, measure the distance between two peaks, called the Period T (seconds), and then use the equation: $f = 1/T$.

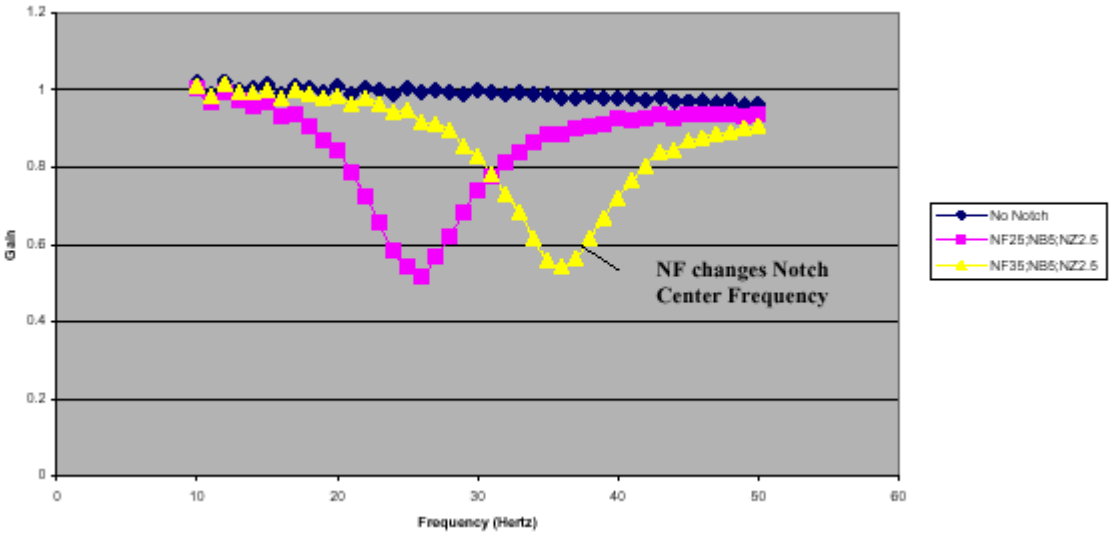


This will be the center frequency for your notch filter, specified as NF. To get the other two parameters, it is easiest to look at an example that shows their relationship to the command output. See the graphs below:




1.2
1
0.8
0.6
0.4
0.2
0
Gain
0
10
20
30
40
50
60
Frequency (Hertz)
NB/NZ ratio controls depth of attenuation
No Notch
NF25;NB5;NZ1
NF25;NB5;NZ2.5


1.6
1.4
1.2
1
0.8
0.6
0.4
0.2
0
Gain
0
10
20
30
40
50
60
Frequency (Hertz)
NZ > NB causes resonance
NZ = NB essentially eliminates Notch
No Notch
NF25;NB5;NZ2.5
NF25;NB5;NZ5
NF25;NB5;NZ7.5

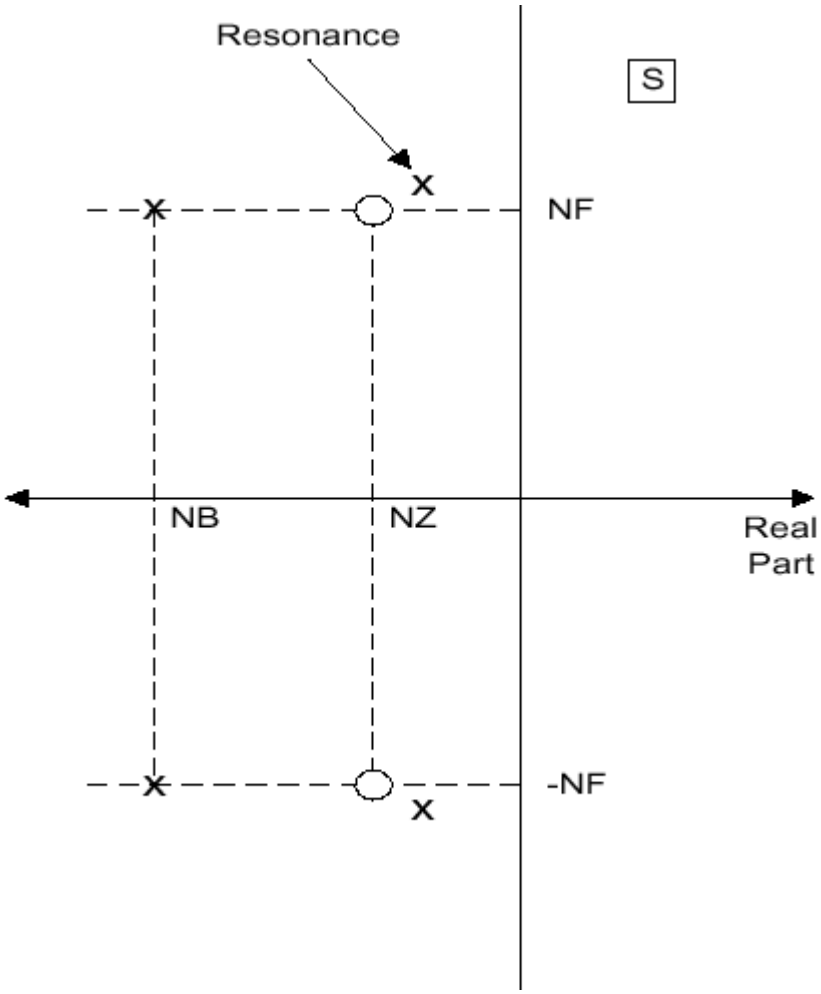


These graphs show how NF, NB, and NZ determine the characteristics of the filter. In particular, NB specifies the bandwidth that is rejected (Figure 1). A larger NB causes a larger range of frequencies to be attenuated. The ratio of NB/NZ controls the amount of attenuation, or depth of the notch (Figure 2). A larger ratio causes a higher amount of attenuation. However, a ratio equal to one should have very little, or no effect, on the output (Figure 3). A ratio greater than one will *amplify* the output signal (Figure 3) causing a resonance. For consistency, these notch waveforms all have a center frequency of 25Hz, except for the last one (Figure 4) which has a NF of 35 and is therefore shifted to the right.

A simple method for attaining your NF,NB, and NZ parameters is the following:

- •Estimate resonance frequency.
- •Set NF to resonance frequency in Hz.
- •Set NB = 1/2 NF.
- •Set NZ between zero and 5.

Although the theory behind a notch filter is beyond the scope of this application note, a general overview may clarify how the notch works. As shown, the notch filter compensates for a resonance in the system. One method of illustrating this is by looking at the poles and zeroes of the transfer function plotted on the *s*-plane.



Resonance shows up as a pair of complex poles with a real part. A notch filter attempts to cancel the unwanted poles by placing zeroes on top of them and placing new poles in a more desirable location. The following diagram shows the pole-zero configuration of a general system with resonance and a notch filter:

A notch filter can be extremely helpful when used properly, however it is not right for every system. Incorrect placement of the Notch can cause system instability, and a notch filter puts extra overhead on the CPU of the controller. A general rule of thumb is to only use a notch when resonance has been found that cannot be eliminated with the controller's standard PID filter of the controller. Also, the notch filter is only effective with a single resonant frequency.

NOTES:

3 Communications

Introduction

The SMC–4000 has one RS232 port and one Ethernet port. The RS-232 is a standard serial link with communication baud rates up to 19.2kbaud. The Ethernet port is a 10Base-T link.

Controller Response to Data

Most SMC–4000 instructions are represented by two characters followed by the appropriate parameters. Each instruction must be terminated by a carriage return or semicolon.

Instructions are sent in ASCII, and the SMC–4000 decodes each ASCII character (one byte) one at a time. It takes approximately 0.5 msec for the controller to decode each command.

After the instruction is decoded, the SMC–4000 returns a colon (:) if the instruction was valid or a question mark (?) if the instruction was not valid or was not recognized.

For instructions requiring data, such at Tell Position (TP), the SMC–4000 will return the data followed by a carriage return, line feed and : .

It is good practice to check for : after each command is sent to prevent errors. An echo function is provided to enable associating the SMC–4000 response with the data sent. The echo is enabled by sending the command EO 1 to the controller.

RS232 Port

The SMC–4000 has a single RS232 connection for sending and receiving commands from a PC or other terminal. The pin connections for the RS232 connection are as follows.

Port 1

1 RTS – input	6 RTS – input
2 Transmit Data - output	7 CTS – output
3 Receive Data - input	8 RTS – input
4 CTS – output	9 No connection
5 Ground	

Configuration

Although Yaskawa's YTerm software automatically configures the port, you may need to manually configure the PC's serial port if using third party software. Configure the computer for 8-bit data, one start-bit, one stop-bit, full duplex and no parity.

Handshaking Modes

The RS232 port is configured for hardware handshaking. In this mode, the RTS and CTS lines are used. The CTS line will go high whenever the SMC–4000 is not ready to receive additional characters. The RTS line will inhibit the SMC–4000 from sending additional characters.

If a device that is used in conjunction with the SMC–4000 does not support hardware handshaking, solder a jumper across the CTS and RTS lines in the cable. Remember that doing so may degrade communication reliability.

SMC Communication Protocol Guidelines

The following items outline details of the simple ASCII communication protocol which the SMC–4000 implements. Note: throughout this section, strings are enclosed in single quotes, and characters are enclosed in greater than / less than signs < >.

1) To query a variable value use the MG command: Example `MG VAR<cr>' where MG is the message command and VAR is a variable defined in the controller.

2) To query a command value use the `MG _TPX<cr>' or `TPX <cr>' where MG is the controller's message command and TP is the command to return the current position. For other details, see the command section of an SMC manual. It describes the possible methods of obtaining data when multiple axes are involved.

3) To set a variable, use `VAR=105<cr>'

4) To se a command parameter, use `PRX=12345<cr>' where PR is the Position Relative command, "X" is the X axis, and the value assigned after the equal sign is the relative move distance specified for the X axis. For other details, see the command section of an SMC manual. It describes the possible methods of obtaining data when multiple axes are involved. Multiple axes can be set at once. Example `PR 12345,6789<cr>' where PR is the Position Relative command, the first value is assigned to the X axis and the second value is assigned to the Y axis. If an axis does not need to be set, it can be omitted as follows: `PR ,,54321<cr>' which will set only the third (Z) axis.

5) Hardware Handshaking is always recommended when communicating with the SMC family of controllers. It is the primary method used by the controller to synchronize communication with external devices. The SMC does not support software handshaking, and simply using three-wire communication will result in possible character loss.

6) When sending a command string of any kind to the controller, verify that the echo is active (EO1) and matches the computer's outgoing string before sending the carriage return. Compare the echo, then either send the carriage return <cr> (if good) or send the backslash character <\> to flush the controller's buffer in the controller (if bad) then resend. Depending on the environment, retry the same string up to 3-5 times before finally determining that there is a serious communication failure.

7) When a message retry is required, send the backslash <\> character to flush the buffer in the SMC so the next command string can be correctly understood. If the buffer contains a partial message, an

additional message could look like a bad message to the SMC, causing another '?<cr><lf>'. Note that it is not neccessary to flush th ebuffer during normal communication.

8) Do not use `VAR=<cr>' to request the SMC to return a variable value. If there was an error in transmission, and the string that the SMC received was not a variable that already exists in the controller, it creates a new variable. If this happens enough times, the controller will fill its variable space. We recommend using `MG VAR<cr>' which is more reliable, meaning if a bad transmission occurs, the SMC will respond with a '?<cr><lf>', and not create an unwanted variable. Note: Use the 'LV<cr>' (List Variables) command to see if there are any erroneous variables in the controller.

9) Use the 'TC<cr>' command to get the error code if a question mark ever appears in a response string.

Ethernet Configuration

Communication Protocols

The Ethernet is a local area network through which information is transferred in units known as packets. Communication protocols are necessary to dictate how these packets are sent and received. The SMC–4000 supports two industry standard protocols, TCP/IP and UDP/IP. The controller will automatically respond in the format in which it is contacted.

TCP/IP is a "connection" protocol. The master must be connected to the slave in order to begin communicating. Each packet sent is acknowledged when received. If no acknowledgement is received, the information is assumed lost and is resent.

Unlike TCP/IP, UDP/IP does not require a connection. This protocol is similar to communicating via RS232. If information is lost, the controller does not return a colon or question mark. Because the protocol does not provide for lost information, the sender must re-send the packet.

Although UDP/IP is more efficient and simple, Yaskawa recommends using the TCP/IP protocol. TCP/IP insures that if a packet is lost or destroyed while in transit, it will be resent.

Ethernet communication transfers information in ‘packets’. The packets must be limited to 470 data bytes or less. Larger packets could cause the controller to lose communication.

***Note To avoid losing information in transit, Yaskawa recommends that the user wait for an acknowledgement of receipt of a packet before sending the next packet.**

***Note A command sent over an Ethernet Telnet session must reside in one packet. This means that a Telnet emulator must not send a command such as MG_TPX<CR> until the carriage return is present; i.e., do not send one character at a time as the user enters them.**

Addressing

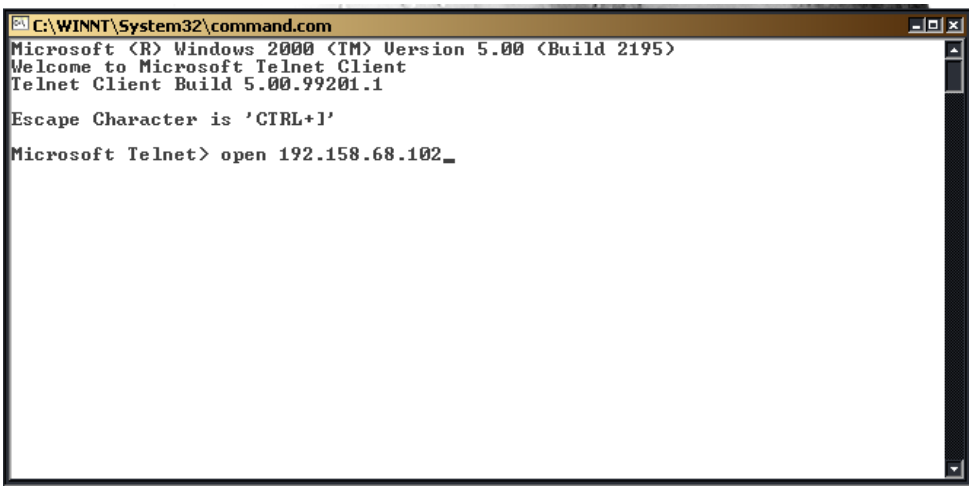
There are three levels of addresses defining Ethernet devices. The first is the Ethernet or hardware address- a unique and permanent 6 byte number, or MAC address. Every device manufactured worldwide has a unique Ethernet address. The SMC–4000 Ethernet address is set by the factory and the last two bytes of the address are the serial number of the controller.

The second level of addressing is the IP address. This is a 32-bit (or 4 byte) number. The IP address is constrained by each local network and must be assigned locally. Assigning an IP address to the controller can be done in a number of ways.

Method #1

The first method is to use the BOOT-P utility via the Ethernet connection (the SMC–4000 must be connected to the network and powered). For an explanation of BOOT-P see *Third Party Software.*

CAUTION: Be sure there is only one BOOT-P server running. If your network has DHCP or BOOT-P running, it may automatically assign an IP address to the controller upon linking it to the network. To ensure that the IP address is correct, please contact your system administrator before connecting the controller to the Ethernet network.



Method #2

The second method for setting an IP address is to send the IA command through the SMC–4000 main RS-232 port. The IP address you want to assign may be entered as a 4 byte number delimited by commas (industry standard uses periods) or a signed 32 bit number. (Ex. IA 124,51,29,31 or IA 2083724575) Type in BN to save the IP address to the controller's non-volatile memory.

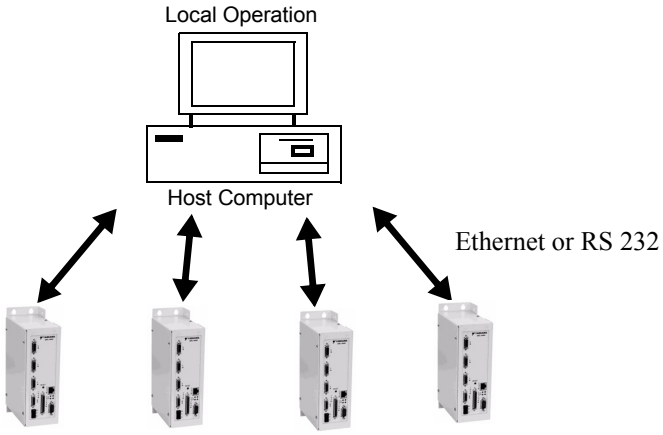
***Note Yaskawa recommends that the IP address selected is not one that can be accessed across the Gateway. The Gateway is an application that controls communication between an internal network and the outside world.**

The third level of Ethernet addressing is the UDP or TCP port number. The Yaskawa controller does not require a specific port number. The port number is established by the master each time it connects to the controller.

Ethernet Handles

An Ethernet handle is a communication resource within a device. The SMC–4000 can have a maximum of 8 Ethernet handles open at any time. Pings and ARPS do not occupy handles. If all 8 handles are in use and a 9th master tries to connect, it will be sent a “reset packet” that generates the appropriate error in its native application.

Each SMC–4000 controls up to four axes of motion, referred to as A-D or XYZ. The host computer can communicate directly with any SMC–4000 via an Ethernet or RS-232 connection.



Ethernet Communication Between Controllers

For this purpose there is the SA command. In its simplest form the SA command is:

SAh=”command string”

Here “command string” will be sent to handle h. For example, the SA command is the means for sending an XQ command to a slave. A more flexible form of the command is:

SAh=field1,field2,field3,field4...field8 Where each field can be a string in quotes or a variable.

For example, to send the command KI,,5,10; assume var1=5 and var2=10 and send the command:

SAF=”KI”,var1,var2

When the master sends an SA command to a sender it is possible for the master to determine the status of the command. The response _IHh4 will return a number of 1 to 4.

1	Waiting for the acknowledgement from the slave.
2	A colon (command accepted) has been received.
3	A question mark (command rejected) has been received.
4	The command timed out.

If a command generates multiple responses (such as the TE command), the values will be stored in _SAh0 through _SAh7. If a field is unused its _SA value will be -2^31.

Handling Communication Errors

If a controller has an application program running and the TCP communication is lost, the #TCPERR routine will automatically execute. See the Special Label Example program in the Example Applications for cases where a command is sent to the lost device.

Modbus Support

The Modbus protocol supports communication between masters and slaves. The masters may be multiple PC’s that send commands to the controller. The slaves are typically peripheral I/O devices that receive commands from the controller.

***Note** **There are numerous ways to reset the controller; hardware reset (push reset button or power-down controller) and software resets (through Ethernet or RS232 by entering RS). The only reset that will not cause the controller to disconnect is a software reset via the Ethernet or RS232.**

When the Yaskawa controller acts as the master, the IH command is used to assign handles and connect to its slaves. The IP address may be entered as a 4 byte number separated with commas (industry standard uses periods) or as a 32 bit number. A port may also be specified, but if not, it will default to 502. The protocol (TCP/IP or UDP/IP) to use must also be designated at this time, otherwise, the controller will not connect to the slave. (Ex. IHB=151,25,255,9<179>2 This will open handle #2 and connect to the I/P address 151.25.255.9, port 179, using TCP/IP).

An additional protocol layer is available for speaking to I/O devices. Modbus is an RS-485 protocol that packages information in binary packets that are sent as part of a TCP/IP packet. In this protocol, each slave has a 1 byte slave address.The SMC–4000 can use a specific slave address or default to the handle number.

Modbus protocol has commands called function codes. The SMC–4000 supports 10 major function codes:

Function Code	Definition
01	Read Coil Status (Read Bits)
02	Read Input Status (Read Bits)
03	Read Holding Registers (Read Words)
04	Read Input Registers (Read Words)
05	Force Single Coil (Write One Bit)
06	Preset Single Register (Write One Word)
07	Read Exception Status (Read Error Code)
15	Force Multiple Coils (Write Multiple Bits)
16	Preset Multiple Registers (Write Words)
17	Report Slave ID

The SMC–4000 provides three levels of Modbus communication.

Level 1

The first level allows the user to create a raw packet and receive raw data. It uses the MBh command with a function code of -1. The command format is:

MBh=-1,len, array[ ] where len is the number of bytes

array [ ] is the array with the data

Level 2

The second level incorporates the Modbus structure. This is necessary for sending configuration and special commands to an I/O device. The formats vary depending on the function code that is called. For more information, refer to Command Reference

Level 3

The third level of Modbus communication uses standard Yaskawa commands. Once the slave has been configured, the commands that may be used are @IN[ ], @AN[ ], SB, CB, OB, and AO. For example, AO 2020,8.2 would tell I/O number 2020 to output 8.2 volts.

If a specific slave address is not necessary, the I/O number to be used can be calculated with the following:

I/O Number= (HandleNum*1000)+((Module-1)*4)+(BitNum)

Where HandleNum is the handle number from 1 (A) to 8 (H). Module is the position of the module in the rack from 1 to 16. BitNum is the I/O point in the module from 1 to 4.

If an explicit slave address is to be used, the equation becomes:

I/O Number=(SlaveAddress*1000)+(HandleNum*1000)+((Module-1)*4)+(Bitnum-1)

Using Third Party Software

Yaskawa supports ARP, BOOT-P, and PING, which are Ethernet utilities. ARP is an application that determines the MAC address of a device at a specific IP address. BOOT-P is an application that determines which devices on the network do not have an IP address and assigns the IP address you have chosen to it. PING is used to check the communication between the device at a specific IP address and the host computer.

The SMC–4000 can communicate with a host computer through any application that can send TCP/IP or UDP/IP packets. A good example of this is Telnet, a utility that comes with most Windows systems.

NOTE: A command sent over an Ethernet Telnet session must reside in one packet. This means that a Telnet emulator must not send a command such as MG_TPX<CR> until the carriage return is present; i.e., do not send one character at a time as the user enters them.

NOTES:

4 Command Reference

Command Description

Each executable instruction is listed in the following section in alphabetical order.

The two letter op-code for each instruction is placed in the upper left corner. Below the op-code is a description of the command and required arguments. As arguments, some commands require actual values to be specified following the instruction. These commands are followed by lower case x, y, z, and w. Values may be specified for any axis separately or any combination of axes. Axis values are separated by commas. Examples of valid x,y, z, w syntax are listed below. For the SMC–4000, the axis designators a,b,c,d are used where x,y,z,w can be used interchangeably with a,b,c,d.

Valid x,y,z,w syntax	Comment
AC x	Specify x only
AC x,y	Specify x and y only
AC x,,z	Specify x and z only
AC x,y,z,w	Specify x,y,z,w
AC ,y	Specify y only
AC ,y,z	Specify y and z
AC ,,z	Specify z only
AC ,,,w	Specify w only
AC x,,,w	Specify x and w only
AC a,,,d	Specify a and d only

Where x, y, z and w are replaced by actual values.

A ? returns the specified value for that axis. For example, AC ?,?,?,?, returns the acceleration of the X,Y,Z and W axes.

Other commands require action on the X,Y,Z or W axis to be specified. These commands are followed by uppercase X,Y,Z or W. Action for a particular axis or any combination is specified by writing X,Y,Z or W. No commas are needed. Valid XYZW syntax is listed below. The SMC–4000 uses ABCD axis designators where XYZW can be used interchangeably with ABCD.

Valid XYZW syntax	Comment
SH X	Servo Here, X only
SH XYW	Servo Here, X,Y and W axes
SH XZW	Servo Here, X,Z and W axes
SH XYZW	Servo Here, X,Y,Z and W axes
SH Y	Servo Here, Y only
SH YZW	Servo Here, Y,Z and W axes
SH Z	Servo Here, Z only
SH	Servo Here, all axes
SH W	Servo Here, W only
SH ZW	Servo Here, Z and W axes
SH AB	Servo Here, A,B axes

Where X,Y,Z and W specify axes.

Command Usage

The usage “Description:” specifies the restrictions on allowable execution. “While Moving” states whether or not the command is valid while the controller is performing a previously defined motion. “In a program” states whether the command may be used as part of a user-defined program. “Command Line” states whether the command may be used from the serial port.

“Can be Interrogated” states whether or not the command can be interrogated by using ? to return the specified value. “Used as an Operand” states whether a command can be used to generate a value for another command or variable (i.e. V=_TTX). “Default Format” defines the format of the value with number of digits before and after the decimal point. Finally, “Default Value” defines the values the instruction’s parameters will have after a Master Reset.

AB (Abort)

[Motion]

DESCRIPTION:

AB (Abort) stops motion instantly without controlled deceleration by freezing the profiler. If there is a program executing, AB also aborts the program unless a 1 argument is specified. The command, AB, will shut off the motors (disable the amplifier) for any axis in which the off-on-error function is enabled (see the OE (Off On Error) command). AB aborts motion on all axes in motion and cannot stop individual axes.

ARGUMENTS: AB n where

n = 0 aborts motion and program

n = 1 aborts motion without aborting program

USAGE:

While Moving	Yes	Default Value	n/a
In a Program	Yes	Default Format	n/a
Command Line	Yes		
Can be Interrogated	Yes		
Used as an Operand	Yes		

OPERAND USAGE:

_AB returns the state of the Abort Input

RELATED COMMANDS:

SH (Servo Here): Turns servos back on if they were shut-off by Abort and OE1.

EXAMPLES:

OE 0,0,0,0	Disable OFF/ON error for all axes
AB	Aborts motion unconditionally, motors remain enabled
OE 1,1,1,1	Enable off-on-error
AB	Shuts off amplifier enable and aborts motion
#A	Label - Start of program
JG 20000	Specify jog speed on X-axis
BGX	Begin jog on X-axis
WT 5000	Wait 5000 msec
AB1	Abort motion without aborting program
WT 5000	Wait 5000 milliseconds
SH	Servo Here
JP #A	Jump to Label A
EN	End of the routine

Hint: Use parameter 1 following AB if you want the motion to be aborted or application program will be aborted.

@ABS (Absolute Value)

[Function]

DESCRIPTION:

@ABS returns the absolute value of a number or variable given in square brackets. Note that the @ABS command is a function, which means that it does not follow the convention of the commands, and does not require the underscore when used as an operand.

ARGUMENTS: @ABS [n] ***where***

n is a number

USAGE:

While Moving	Yes	Minimum n value	-2147483647.9999
In a Program	Yes	Maximum n value	2147483647.9999
Not in a program	Yes	Default Value	---
Can be Interrogated	No	Default Format	---
Used as an Operand	Yes		

EXAMPLES:

OE 0,0,0,0	Program TEST
VAR1=-45.6	Set a variable equal to -45.6
MG @ABS[VAR1]	Display the absolute value of VAR1
VAR2=@ABS[VAR1]+100.404	Perform calculation
EN	End of program

AC (Acceleration)

[Motion]

DESCRIPTION:

The Acceleration (AC) command sets the linear acceleration rate for independent moves, such as PR (Position Relative), PA (Position Absolute) and JG (Jog) moves. The parameters input will be rounded down to the nearest factor of 1024. The units of the parameters are counts per second squared. The acceleration rate may be changed during motion. The DC (Deceleration) command is used to specify the deceleration rate.

ARGUMENTS: AC x, y, z, w or ACX=x or AC a, b, c, d where

x, y z, w, or a, b, c, d are unsigned integers

USAGE:

While Moving	Yes	Minimum Value	1024
In a Program	Yes	Maximum Value	67107840
Command Line	Yes	Default Value	256000
Can be Interrogated	Yes	Default Format	8.0
Used as an Operand	Yes		
Virtual Axis	Yes		

OPERAND USAGE:

_ACn contains the value of acceleration in counts/sec^2 where n is an axis letter.

RELATED COMMANDS:

DC (Deceleration)

FA (Acceleration Feedforward)

IT (Independent Time Constant)

EXAMPLES:

AC 150000	Set acceleration to 150000 counts/sec^2
MG_ACX	Request the current acceleration setting
0149504	Returned Acceleration (resolution, 1024)
V=_ACX	Assigns the current acceleration setting to the variable V

HINTS: Specify realistic acceleration rates based on your physical system such as motor torque rating, loads, and amplifier current rating. Specifying an excessive acceleration will cause large following error during acceleration and the motor will not follow the commanded profile. The acceleration feedforward command FA will help minimize error during acceleration.

@ACOS (Arc Cosine)

[Function]

DESCRIPTION:

@ACOS returns the arc cosine, in degrees, of a number or variable which is inserted in square brackets. Note that the @ACOS command is a function, which means that it does not follow the convention of other commands and does not require the underscore when used as an operand.

ARGUMENTS: @ACOS [n] ***where***

n is a number

USAGE:

While Moving	Yes	Minimum n value	-1
In a Program	Yes	Maximum n value	1
Not in a program	Yes	Default Value	---
Can be Interrogated	No	Default Format	---
Used as an Operand	Yes		

EXAMPLES:

#TEST	Program TEST
VAR1=.707	Set a variable equal to .707
MG @ACOS[VAR1]	Display the absolute value of VAR1
VAR2=@ACOS[VAR1]+100.404	Perform calculation
EN	End of program

AD (After Distance)

[Trippoint]

DESCRIPTION:

The After Distance (AD) command is a trippoint used to control the timing of events. This command will hold up the execution of the following command until one of the following conditions have been met:

1. The commanded motor position crosses the specified relative distance from the start of the move.
2. The motion profiling on the axis is complete.
3. The commanded motion is in the direction which moves away from the specified position.

The units of the command are quadrature counts. The motion profiler must be on or the trippoint will automatically be satisfied.

ARGUMENTS: ADx, y, z, w or ADX=x or AD a, b, c, d where

x, y z, w, or a, b, c, d are unsigned integers

USAGE:

While Moving	Yes	Minimum Value	0
In a Program	Yes	Maximum Value	2147483647
Command Line	Yes	Default Value	---
Can be Interrogated	No	Default Format	---
Used as an Operand	No		
Virtual Axis	No		

RELATED COMMANDS:

AR (After Relative)

AP (After Absolute Position)

EXAMPLES:

#A;DP0	Begin Program
PR 10000	Specify position
BG	Begin motion
AD 5000	Wait until profiler passes 5000 units from start of move.
MG "Halfway" ;TP	Send message
EN	End Program

Hint: The AD command is accurate to the number of counts that occur in 2 msec. Multiply speed by 2 msec to obtain the maximum position error in counts. Remember AD measures incremental distance from start of move on one axis.

AE (Absolute Encoder)

[Function]

DESCRIPTION:

The AE (Absolute Encoder) command reads the encoder data fom a Yaskawa absolute encoder. The command automatically enters the absolute position received in the axes' position register. The motor must be off (MO) befire using this command.

ARGUMENTS: AEX=32768 AE x, y, z, w AE a,b, c, d Where

x, y, and z are signed numbers (encoder resolution, post quadrature).

These values are the number of encoder pulses per revolution after quadrature.

Hint: If the rotation direction bit in the servo amplifier is set for reverse rotation, use a negative number with this command.

USAGE:

While Moving	No	Defualt Value	–
In a Program	Yes	Default Format	–
Not in Program	Yes	Minimum Value	1
Can be Interrogated	No	Maximum Value	16777216
Used in an Operand	Yes		

OPERAND USAGE:

_AE returns the number of the last encoder that was read. 0= X axis, 1 = Y Axis, ...

RELATED COMMANDS:

TY (Tell Yaskawa Absolute Encoder)

QY (Query Yaskawa Absolute Encoder Alarm)

EXAMPLES:

MOX	Motor Off
AEX=4096	Read X absolute encoder
DPX=_TPX+XhmOfs	Add offfset variable to current absolute position
TPX	Tell position of X axis
SHX	Enable X servo amplifier

Hint: A command error will be generated if an absolute encoder fails to respond. If the #CMDERR special label is used, a routine can be written to display the type of encoder problem using the "QY and "_AE" commands. See the absolute encoder section in the options chapter for more information.

AI (After Input)

[Trippoint]

DESCRIPTION:

The AI command is used in motion programs to wait until after the specified input condition has occurred. If n is positive, it waits for the input to go high. If n is negative, it waits for input to go low. To wait for a transition from high to low or low to high, put two AI commands together. AI is only available for local inputs. The AI command causes the corresponding program thread to go dormant until the specified logic state is true. If the program thread must check other logic, use the JP (Jump to Program Location) or IF–JS (Jump to Subroutine) commands instead.

ARGUMENTS: AI +/-n where

n is a signed integer

USAGE:

While Moving	Yes	Minimum Value	1
In a Program	Yes	Maximum Value	8
Command Line	Yes	Default Value	---
Can be Interrogated	No	Default Format	---
Used as an Operand	No		

RELATED COMMANDS:

@IN (Input): Function to read digital input 1 through 8

II (Input Interrupt)

#ININT: Special label for input interrupt

EXAMPLES:

#A	Begin Program
AI 7	Wait until input 7 is high
SP 10000	Speed is 10000 counts/sec
AC 20000	Acceleration is 20000 counts/sec2
PR 400	Specify position
BG	Begin motion
AI+ 7; AI- 7	Wait for falling edge on input 7
EN	End Program

AL (Arm Latch)

[Setting]

DESCRIPTION:

The AL command enables the latching function (high speed position capture) of the controller. When the AL command is used to arm the position latch, the encoder position of the main encoder input will be captured when the corresponding input is in the "active" state.. When interrogated or used in an operand the AL command will return a 1 if the latch is armed or a zero after the latch has occurred. The command RL (Report Latch) returns the captured position value. The CN (Configure Limit Switches) command will change the active state of the latch.

ARGUMENTS: ALn ***where***

n = XYZW or ABCD for the main encoder latch

USAGE:

While Moving	Yes	Minimum Value	n/a
In a Program	Yes	Maximum Value	n/a
Command Line	Yes	Default Value	n/a
Can be Interrogated	No	Default Format	n/a
Used as an Operand	Yes		
Virtual Axis	No		

OPERAND USAGE:

_ALn contains the state of the latche (main & auxiliary) where n is an axis letter. 0 = not armed, 1 = armed.

RELATED COMMANDS:

RL (Report Latch)

CN (Configure Limit Switches)

EXAMPLES:

#START	Start program
ALX	Arm latch on X axis
JG 50000	Set up jog at 50000 counts/sec
BG	Begin the move
#LOOP	Loop until latch has occurred
JP #LOOP,_ALX=1	
RL	Transmit the latched position
EN	End of program

AM (After Motion)

[Trippoint]

DESCRIPTION:

The AM command is a trippoint used to control timing of events. This command holds up execution of the following commands until the current move on the specified axis or axes is completed. AM occurs when the profiler is finished generating the last position command. However, the servo motor may not be in final position. Use TE (Tell Error) to verify position error for servos or use the MC (Motion Complete) trippoint to wait until final position is reached by the servo.

ARGUMENTS: AM XYZWS or ABCD where

X, Y, Z, W or A, B, C, D are axis designators. S indicates an interpolation sequence. No argument specifies that motion on all axes must be complete.

USAGE:

While Moving	Yes	Minimum Value	n/a
In a Program	Yes	Maximum Value	n/a
Command Line	Yes	Default Value	---
Can be Interrogated	No	Default Format	---
Used as an Operand	No		
Virtual Axis	No, use _BGN		

RELATED COMMANDS:

BG (Begin) : _BG returns a 0 if motion complete

MC (Motion Complete)

TW (Time Wait)

EXAMPLES:

#MOVE	Start of program
PR 5000	Position relative move
BG	Begin motion
AM	Wait until motion is complete
EN	End of Program
#F;DP 0	Program F
PR 5000	Position relative move
BG	Begin motion
AM	Wait until motion is complete
MG "DONE";TP	Print message
EN	End of Program

HINT: ***AM command controls the timing between multiple move sequences. If the motor is in the middle of a position relative move (PR), a position absolute move (PA, BG) cannot be made until the first move is complete. Use AM to pause the program sequences until the first motion is complete. AM tests for profile completion. Another testing method is to query the operand, _BG. This is equal to 1 during motion, and 0 when motion profiling is complete.***

AO (Analog Out)

[I/O]

DESCRIPTION:

The AO command sets the analog output voltage of the ModBus devices connected via Ethernet. This command is an alternative to using MB (Modbus).

ARGUMENTS: AO m, n where

m = (SlaveAddress*1000) + (HandleNum*1000) + (Module-1)*4) + (Bitnum-1)

Slave Address is used when the ModBus device has slave devices connected to it and specified as Addresses 0 to 225.

HandleNum is the handle specifier from A to H (1 - 8).

Module is the position of the module in the rack from 1 to 16.

BitNum is the I/O point in the module from 1 to 4.

n = the voltage which ranges from 9.9982 to –9.9982.

USAGE:

While Moving	Yes	Minimum n Value	-9.9982
In a Program	Yes	Maximum n Value	9.9982
Command Line	Yes	Default Value	n/a
Can be Interrogated	No	Default Format	---
Used as Operand	No		

OPERAND USAGE: No

RELATED COMMANDS:

MB (Modbus)

EXAMPLES:

AO 6016, 8.2	Sets analog output on modbus device on handle F to 8.2V

AP (After Absolute Position)

[Trippoint]

DESCRIPTION:

The After Position (AP) command is a trippoint used to control the timing of events. This command will hold up the execution of the following command until one of the following conditions have been met:

1. The commanded motor position crosses the specified absolute position.
2. The motion profiling on the axis is complete.
3. The commanded motion is moving away from the specified position.

The units of the command are quadrature counts. The motion profiler must be active or the trippoint will automatically be satisfied.

ARGUMENTS: AP x, y, z, w or APX=x or AP a, b, c, d where

x, y z, w, or a, b, c, d are unsigned integers

USAGE:

While Moving	Yes	Minimum Value	-2147483648
In a Program	Yes	Maximum Value	2147483647
Command Line	Yes	Default Value	---
Can be Interrogated	No	Default Format	---
Used as an Operand	No		
Virtual Axis	No		

RELATED COMMANDS:

AR (After Relative)

AD (After Distance)

MF (Motion Forward)

EXAMPLES:

#TEST	Program B
DP0	Define position as zero
JG 1000	Set jog with speed of 1000 counts/sec
BG	Begin move
AP 2000	After passing position 2000
V1=_TP	Assign V1 the Xaxis X position
MG "Position is", V1	Print Message
ST	Stop axis
EN	End of Program

HINT: ***The accuracy of the AP command is the number of counts that occur in 2 msec. Multiply the speed by 2 msec to obtain the maximum error. AP tests for absolute position. Use the AD (After Distance) command to measure incremental distances.***

AR (After Relative)

[Trippoint]

DESCRIPTION:

The After Relative (AR) command is a trippoint used to control the timing of events. This command will hold up the execution of the following command until one of the following conditions have been met:

1. The commanded motor position crosses the specified relative distance from either the start of the move or the last AR or AD command.
2. The motion profiling on the axis is complete.
3. The commanded motion is in the direction which moves away from the specified position.

The units of the command are quadrature counts. The motion profiler must be active or the trippoint will automatically be satisfied.

ARGUMENTS: AR x, y, z, w or ARX=x or AR a, b, c, d where

x, y z, w, or a, b, c, d are unsigned integers

USAGE:

While Moving	Yes	Minimum Value	0
In a Program	Yes	Maximum Value	2147483647
Command Line	Yes	Default Value	---
Can be Interrogated	No	Default Format	---
Used as an Operand	No		
Virtual Axis	No		

RELATED COMMANDS:

AD (After Distance)

AP (After Absolute Position)

EXAMPLES:

#A;DP 0	Begin Program
JG 50000	Specify jog speed
BG	Begin motion
#B	Label
AR 5000	After passing 5000 counts of relative distance on X-axis from the last trippoint
MG "Passed_X";TP	Send message
JP #B	Jump to Label #B
EN	End Program

HINT: AR is used to specify incremental distance from last AR or AD command. Use AR if multiple position trippoints are needed in a single motion sequence.

AS (At Speed)

[Trippoint]

DESCRIPTION:

The AS command is a trippoint that occurs when the generated motion profile has reached the specified speed. This command will hold up execution of the following command until the speed is reached. The AS command will operate after either accelerating or decelerating. If the commanded speed is not reached, the trippoint will be triggered after the motion is stopped (after deceleration).

ARGUMENTS: AS XYZWS or ABCD where

X, Y, Z, W or A, B, C, D are axis designators. S indicates an interpolation sequence. No argument specifies that motion on all axes is complete.

USAGE:

While Moving	Yes	Minimum Value	n/a
In a Program	Yes	Maximum Value	n/a
Command Line	Yes	Default Value	---
Can be Interrogated	No	Default Format	---
Used as an Operand	No		
Virtual Axis	No		

EXAMPLES:

#SPEED	Program A
PR 100000	Specify relative position
SP 10000	Specify speed
BG	Begin motion
AS	Wait until after commanded speed is reached
MG "At Speed"	Print Message
EN	End of Program

WARNING: The AS command applies to a trapezoidal velocity profile only with linear acceleration. AS used with S-curve profiling may be inaccurate.

@ASIN (Arc Sine)

[Function]

DESCRIPTION:

@ASIN returns the arc sine, in degrees, of a number or variable which is inserted in square brackets. Note that the @ASIN command is a function, which means that it does not follow the convention of other commands, and does not require the underscore when used as an operand.

ARGUMENTS: @ASIN [n] where

n is an unsigned integer

USAGE:

While Moving	Yes	Minimum n value	-1
In a Program	Yes	Maximum n value	1
Not in a program	Yes	Default Value	---
Can be Interrogated	No	Default Format	---
Used as an Operand	Yes		

EXAMPLES:

#TEST	Program TEST
MG @ASIN[VAR1]	Set variable
VAR1=.707	Display the arc sine of .707
VAR2=@ASIN[VAR1]+5	Perform calculation
EN	End of program

AT (After Time)

[Trippoint]

DESCRIPTION:

The AT command is a trippoint which is used to hold up execution of the next command until after the specified time has elapsed. The time is measured with respect to a defined reference time. AT 0 establishes the initial reference. AT n specifies n msec from the reference. AT -n specifies n msec from the reference and establishes a new reference after the elapsed time period. This command is useful for waiting an accurate amount of time duration while still being able to perform some other operations as long as they require less time than the AT time.

ARGUMENTS: AT n where

n is a signed integer

n = 0 defines a reference time at current time

positive n waits n msec from reference

negative n waits n msec from reference and sets new reference after elapsed time period

(AT -n is equivalent to AT n; AT 0)

USAGE:

While Moving	Yes	Minimum Value	-2147483648
In a Program	Yes	Maximum Value	2147483647
Command Line	Yes	Default Value	---
Can be Interrogated	No	Default Format	---
Used as an Operand	No		

EXAMPLES:

The following commands are sent sequentially:

AT 0	Establishes reference time 0 as current time
AT 50	Waits 50 msec from reference 0
AT 100	Waits 100 msec from reference 0
AT -150	Waits 150 msec from reference 0 and sets new reference at 150
AT 80	Waits 80 msec from new reference (total elapsed time is 230 msec)

@ATAN (Arc Tangent)

[Function]

DESCRIPTION:

@ATAN returns the arc tangent, in degrees between 90 and -90, of a number or variable which is inserted in square brackets. Note that the @ATAN command is a function, which means that it does not follow the convention of other commands, and does not require the underscore when used as an operand.

ARGUMENTS: @ATAN [n] where

n is an unsigned integer

USAGE:

While Moving	Yes	Minimum n value	-2147483647
In a Program	Yes	Maximum n value	2147483647
Not in a program	Yes	Default Value	---
Can be Interrogated	No	Default Format	---
Used as an Operand	Yes		

EXAMPLES:

#TEST	Program TEST
MG @ATAN[VAR1]	Display the arc tangent of .707
VAR1=.707	Set variable
VAR2=@ATAN[VAR1]+5	Perform calculation
EN	End of program

AV (After Vector Distance)

[Trippoint]

DESCRIPTION:

The AV command is a trippoint which is used to hold up execution of the next command during coordinated moves such as VP (Vector Position), CR (Circle) or LI (Linear Interpolation Distance). This trippoint occurs when the path distance of a sequence reaches the specified value. The distance is measured from the start of a coordinated move sequence or from the last AV command. The units of the command are quadrature counts.

ARGUMENTS: AVs,t where

s and t are unsigned integers in the range 0 to 2147483647 decimal. 's' represents the vector distance to be executed in the S coordinate system and 't' represents the vector distance to be executed in the T coordinate system.

USAGE:

While Moving	Yes	Default Value	0
In a Program	Yes	Default Format	---
Command Line	Yes		

OPERAND USAGE:

_AVS contains the vector distance from the start of the sequence in the S coordinate system and _AVT contains the vector distance from the start of the sequence in the T coordinate system.

EXAMPLES:

#MOVE;DP 0,0	Label
CAT	Specify the T coordinate system
LMAB	Linear move for A,B
LI 1000,2000	Specify distance
LI 2000,3000	Specify distance
LE	
BGT	Begin motion in the T coordinate system
AV ,500	After path distance = 500,
MG "Path>500";TPAB	Print Message
EN	End Program

HINT: ***Vector Distance is calculated as the square root of the sum of the squared distance for each axis in the linear or vector mode.***

BG (Begin)

[Motion]

DESCRIPTION:

The BG command starts motion. When used as an operand, the BG command will return a 1 if there is a commanded motion in progress, a 0 otherwise. The BG command will result in a command error if a move is already in progress, the servo is not enabled or a limit switch is preventing motion.

ARGUMENTS: BG XYZWST or ABCD where

X, Y, Z, W, N, S, T or A, B, C, D specify the axis or sequence. No argument specifies that motion on all axes must be complete.

USAGE:

While Moving	Yes	Default Value	---
In a Program	Yes	Default Format	---
Command Line	Yes		
Can be Interrogated	No		
Used as an Operand	Yes		
Virtual Axis	Yes		

OPERAND USAGE:

_BGn contains a ‘0’ if motion complete on the axis, otherwise contains a ‘1’ where n is an axis letter.

RELATED COMMANDS:

AM (After Motion)

ST (Stop)

EXAMPLES:

PR 2000	Set up for a relative move
BG	Begin motion
AM	Wait until motion is complete
HM	Issue homing command
BG	Begin motion
AM	Wait until motion is complete
JG 1000	Issue jog command
BG	Begin motion
STATE=_BGX	Assign a 1 to STATE if the axis is performing a move

HINT: ***You cannot give another BG command until current BG motion has been completed. Use the AM trippoint to wait for motion complete between moves. Another method for checking motion complete is to test for _BG being equal to 0.***

BK (Breakpoint)

[Configuration]

DESCRIPTION:

For debugging. Causes the controller to pause execution of the given thread at the the given program line number (which is not executed). All other threads continue running. Only one breakpoint may be armed at any time. After a breakpoint is encountered a new breakpoint can be armed (to continue execution to the new breakpoint) or BK will resume program execution. The SL (Single Step) command can be used to single step from the breakpoint. The breakpoint can be armed before or during thread execution.

ARGUMENTS: BK n, m where

n is an integer in the range of 0 to 1999, which is the line number at which to stop/ n must be a valid line number in the chosen thread.

m is an integer in the range of 0 to 7. It designates the chosen thread.

USAGE:

While Moving	Yes	Default Value of m	0
In a Program	No		
Command Line	Yes		

OPERAND USAGE:

_BK tells whether a breakpoint has been armed, whether it has been encountered, and the program line number of the breakpoint.

= -LineNumber: Breakpoint armed

= LineNumber: Breakpoint encounterred

= -2147483648: Breakpoint not armed

RELATED COMMANDS:

SL (Single Step)

TR (Trace Mode)

EXAMPLES:

BK3	Pause at line 3 (the 4th line) in thread 0
BK 5	Continue at line 5
SL	Execute the next line
SL 3	Execute the next 3 lines
BK	Resume normal execution

BL (Backward Limit)

[Setting]

DESCRIPTION:

The BL command sets the reverse software limit. If this limit is exceeded during a commanded motion, the motion will decelerate to a stop. Reverse motion beyond this limit is not permitted. The reverse limit is activated at position n-1 count. To disable the reverse limit, set n to -2147483648. The units are in quadrature counts.

ARGUMENTS: BLx, y, z, w or BLX=x or BL a, b, c, d where

x, y z, w, or a, b, c, d are signed integers

-2147483648 turns off the reverse limit.

USAGE:

While Moving	Yes	Minimum Value	-2147483648
In a Program	Yes	Maximum Value	2147483647
Command Line	Yes	Default Value	-2147483648
Can be Interrogated	Yes	Default Format	Position format
Used as an Operand	Yes		
Virtual Axis	No		

OPERAND USAGE:

_BLn contains the value of the reverse software limit where n is an axis letter.

RELATED COMMANDS:

EXAMPLES:

"FL"	Forward Limit
"PF"	Position Formatting
#TEST	Test Program
AC 1000000	Set Acceleration Rate
DC 1000000	Set Deceleration Rate
BL -15000	Set Reverse Limit
JG -5000	Jog Reverse
BG	Begin Motion
AM	After Motion (soft limit occurred)
TP	Tell Position
EN	End Program

BN (Burn Parameters)

[General]

DESCRIPTION:

The BN command saves certain controller parameters in non-volatile EEPROM memory. This command takes approximately one second to execute and must not be interrupted. If the burn is disrupted by power failure, a memory checksum error will result. The controller returns a <:> when the Burn is complete.

PARAMETERS SAVED DURING BURN:

AC	EP	KP	PF
BL	ER	LZ	SB
CE	ET (table)		SP
CF	FA	MO (MOTOR OFF or ON)	TL
CM	FL	MT	TM
CN	GA	NA	TR
CW	GR	NB	VA
DC	IA	NF	VD
DV	IL	NZ	VF
EA	IT	OE	VS
EM	KD	OF	VT
EO	KI	OP	

ARGUMENTS: None

OPERAND USAGE: _BN returns the serial number of the controller.

USAGE:

While Moving	Yes	Default Value	---
In a Program	Yes	Default Format	---
Command Line	Yes		
Can be Interrogated	No		
Used as an Operand	Yes		

BP (Burn Program)

[General]

DESCRIPTION:

The BP command saves the application program and cam data in non-volatile EEPROM memory. This command typically takes up to 10 seconds to execute and must not be interrupted. If the burn is disrupted by power failure, a memory checksum error will result. The controller returns a <: >when the Burn is complete.

ARGUMENTS: None

USAGE:

While Moving	No	Default Value	---
In a Program	No		
Command Line	Yes		
Can be Interrogated	No		
Used as an Operand	No		

BV (Burn Variables)

[General]

DESCRIPTION:

The BV command saves the defined variables and arrays in non-volatile EEPROM memory. This command typically takes up to 2 seconds to execute and must not be interrupted. **If the burn is disrupted by power failure, a memory checksum error will result.** The controller returns a <:> when the Burn variables are complete.

ARGUMENTS: None

USAGE:

While Moving	No	Default Value	---
In a Program	No		
Command Line	Yes		
Can be Interrogated	No		
Used as an Operand	Yes		

Note: A checksum error can only be cleared by performing a master reset, which is the ASCII characters CTRL R and CTRL S snt over the serial port.

CA (Coordinate Axes)

DESCRIPTION:

The CA command specifies the coordinate system to apply proceeding vector commands. The following commands apply to the active coordinate system as set by the CA command.

CR	ES	LE	LI	LM
TN	VE	VM	VP	

ARGUMENTS: CAS or CAT where

CAS specifies that proceeding vector commands shall apply to the S coordinate system

CAT specifies that proceeding vector commands shall apply to the T coordinate system

OPERAND USAGE:

_CA contains a 0 if the S coordinate system is active and a 1 if the T coordinate system is active.

USAGE:

While Moving	Yes	Default Value	CAS
In a Program	Yes	Default Format	---
Command Line	Yes		

RELATED COMMANDS:

VP (Vector Position)

VS (Vector Speed)

VD (Vector Deceleration)

VA (Vector Acceleration)

VM (Coordinated Motion Mode)

VE (Vector Sequence End)

BG (Begin)

EXAMPLES:

CAT	Specify T coordinate system
VMAB	Specify vector motion in the A and B plane
VS 10000	Specify vector speed
CR 1000,0,360	Generate circle with radius of 1000 counts, start at 0 degrees and complete one circle in counterclockwise direction.
VE	End Sequence
BGT	Start motion of T coordinate system

CB (Clear Bit)

[I/O]

DESCRIPTION:

The CB command clears a bit on the output port by setting it to logic zero. Modbus outputs can be cleared also.

ARGUMENTS: CB n where

n is an integer corresponding to a specific output on the controller to be cleared (set to 0). The first output on the controller is denoted as output 1. An SMC–4000 controller has 8 digital outputs plus applicable I/O connected by Modbus.

MODBUS:

Note: When using Modbus devices, the I/O points of the modbus devices are calculated using the following formula:

n = (SlaveAddress*1000) + (HandleNum*1000) + ((Module-1)*4) + (Bitnum-1)

Slave Address is used when the ModBus device has slave devices connected to it and specified as Addresses 0 to 255. The use of slave devices for modbus are very rare and this number will usually be 0.

HandleNum is the handle specifier from A to H (1 - 8).

Module is the position of the module in the rack from 1 to 16.

BitNum is the I/O point in the module from 1 to 4.

USAGE:

While Moving	Yes	Default Value	---
In a Program	Yes	Default Format	---
Command Line	Yes		
Can be Interrogated	No		
Used as an Operand	No		

RELATED COMMANDS:

SB (Set Bit)

OP (Output Port)

EXAMPLES:

CB 1	Clear output bit 1
CB 2	Clear output bit 2
CB 3	Clear output bit 3
CB 6002	Clear output 2 on Modbus device on handle F

CD (Contour Data)

[Motion]

DESCRIPTION:

The CD command specifies the incremental position for an arbitrary motion profile. The units of the command are in quadrature counts. This command is only applicable in CM (Contour Mode).

ARGUMENTS: CD x, y, z, w or CDX=x or CD a, b, c, d where

x, y z, w, or a, b, c, d are signed integers

USAGE:

While Moving	Yes	Minimum Value	-32767
In a Program	Yes	Maximum Value	+32767
Command Line	Yes	Default Value	0
Can be Interrogated	No	Default Format	---
Used as an Operand	No		
Virtual Axis	No		

RELATED COMMANDS:

CM (Contour Mode)

WC (Wait for Contour)

DT (Delta Time)

EXAMPLES:

CM	Specify Contour Mode
DT 4	Specify time increment for contour mode
CD 200	Specify incremental positions of 200 counts
WC	Wait for complete
CD 100	New position data
WC	Wait for complete
DT0	Stop Contour
CD 0	Exit Mode

CE (Configure Encoder)

[Configuration]

DESCRIPTION:

The CE command configures the encoder inputs to the quadrature type or the pulse and direction type. It also allows inverting the polarity. The configuration applies independently to the main axis encoder and the auxiliary encoder inputs.

Warning: This command interacts with the MT command, which specifies motor type and direction. Use caution (motor off, machine estopped) when changing the MT or CE commands. If the two commands are not in agreement with each other, the motor will run away at full speed when enabled.

ARGUMENTS: CE x, y, z, w or CEX=x or CE a, b, c, d where

x, y z, w, or a, b, c, d are unsigned integers

Each integer is the sum of two integers r and s which configure the main and the auxiliary encoders according to the chart below.

R =	MAIN ENCODER TYPE	S =	AUXILIARY ENCODER TYPE
0	Normal quadrature	0	Normal quadrature
1	Normal pulse and direction	4	Normal pulse and direction
2	Reversed quadrature	8	Reversed quadrature
3	Reversed pulse and direction	12	Reversed pulse and direction

For example: CEX = 10 implies r = 2 and s = 8, both encoders are reversed quadrature.

USAGE:

While Moving	Yes	Minimum Value	0
In a Program	Yes	Maximum Value	10
Command Line	Yes	Default Value	0
Can be Interrogated	Yes	Default Format	2.0
Used as an Operand	Yes		
Virtual Axis	No		

OPERAND USAGE:

_CEn contains the value of encoder type for the main and auxiliary encoder where n is an axis letter.

RELATED COMMMANDS:

MT (Motor Type)

EXAMPLES:

CE 0	Configure encoders
MG_CEX	Interrogate configuration
V = _CE	Assign configuration to a variable

CF (Configure Messages)

[Configuration]

DESCRIPTION:

Sets the controller's default port for unsolicited messages. By default, the SMC–4000 controller will send unsolicited responses to the RS-232 serial port. An unsolicited message is one generated in the controller, i.e.; a program fault message or a message resulting from the MG (Message) command with no port designation specified. Messages originating in the controller can be forced to a specific port– See the MG (Message) command.

ARGUMENTS: ***CF n*** ***where***

n is A through H for Ethernet handles 1 thru 8, S for serial port.

USAGE:

While Moving	Yes	Default Value	83 ("S")
In a Program	Yes	Default Format	Decimal representation
Command Line	Yes		

OPERAND USAGE:

_CF will return the current port selected for unsolicited responses from the controller. The _CF will return a decimal value of the ASCII code.

EXAMPLES:

CFA	Select Ethernet handle A to return unsolicited responses.
MG_CF	Interrogate configuration
:65.000	Response from _CF showing handle A as default port. 65 is the ASCII value for "A".

CM (Contour Mode)

[Setting]

DESCRIPTION:

The Contour Mode is initiated by the instruction CM. This mode allows the generation of an arbitrary motion trajectory. The CD command specifies the position increment, and the DT command specifies the time interval.

The CM? or _CM commands can be used to check the status of the Contour Buffer. A value of 1 returned indicates that the Contour Buffer is full. A value of 0 indicates that the Contour Buffer is empty.

ARGUMENTS: CM XYZW or ABCD

USAGE:

While Moving	No	Default Value	---
In a Program	No	Default Format	1.0
Command Line	Yes		
Can be Interrogated	No		
Used as an Operand	Yes		
Virtual Axis	No		

OPERAND USAGE:

_CM contains a ‘0’ if the contour buffer is empty, otherwise contains a ‘1’ meaning the buffer is full.

RELATED COMMANDS:

CD (Contour Data)

WC (Wait for Contour)

DT (Delta Time)

EXAMPLES:

V=_CM; MGV	Return Contour Buffer Status
1	Contour Buffer is full
CM	Specify Contour Mode

CN (Configure Limit Switches)

[Configuration]

DESCRIPTION:

The CN command configures the polarity of the limit switches, the home switch and the latch input.

ARGUMENTS: CN m,n,o,p where

m, n, o, p are integers.

m =	1	Limit switches active high
	-1	Limit switches active low
n =	1	Home switch configured to drive motor in forward direction when input is high upon initial HM execution. See HM and FE commands
	-1	Home switch configured to drive motor in reverse direction when input is high upon initial HM execution. See HM and FE commands
o =	1 *	Latch input is active high
	-1	Latch input is active low
p =	1	Selective abort function (See chart below)
	0	Normal input operation.

***Note: The latch function will occur within 25 µsec only when used in active low mode, the opto isolator requires more time if active high.**

SELECTIVE ABORT

Input	Axis
5	X
6	Y
7	Z
8	W

USAGE:

While Moving	Yes	Default Value	-1, -1, -1, 0
In a Program	Yes	Default Format	2.0
Command Line	Yes		
Can be Interrogated	No		
Used as an Operand	Yes		

OPERAND USAGE:

_CN0 Contains the limit switch configuration.

_CN1 Contains the home switch configuration.

_CN2 Contains the latch input configuration.

_CN3 Contains status of selective abort function.

RELATED COMMANDS:

MT (Motor Type)

EXAMPLES:

CN 1,1	Sets limit and home switches to active high
CN, -1	Sets input latch active low
MG_CN1	Returns Home input configuration
MG_CN2	Returns Latch input configuration

@COM (2's Complement)

[Function]

DESCRIPTION:

@COM returns the complement of a number or variable which is inserted in square brackets. Note that the @COM command is a function, which means that it does not follow the convention of other commands, and does not require the underscore when used as an operand.

ARGUMENTS: @COM [n] ***where***

n is a number

USAGE:

While Moving	Yes	Minimum n value	-2147483647.9999
In a Program	Yes	Maximum n value	2147483647.9999
Not in a program	Yes	Default Value	---
Can be Interrogated	No	Default Format	---
Used as an Operand	Yes		

EXAMPLES:

#TEST	Program TEST
VAR1=1234	Set variable
MG @COM[VAR1]	Display the complement of 1234
VAR2=@COM[VAR1]+99	Perform calculation
EN	End of program

@COS (Cosine)

[Function]

DESCRIPTION:

@COS returns the cosine of a number or variable given in square brackets using units of degrees. Note that the @COS command is a function, which means that it does not follow the convention of the commands, and does not require the underscore when used as an operand.

ARGUMENTS: @COS [n] where

n is a number

USAGE:

While Moving	Yes	Minimum n value	-32768
In a Program	Yes	Maximum n value	32768
Not in a program	Yes	Default Value	---
Can be Interrogated	No	Default Format	---
Used as an Operand	Yes		

EXAMPLES:

#TEST	Program TEST
VAR1=60	Set variable
MG @COS[VAR1]	Display the value of the sine of VAR1
VAR2=@COS[VAR1]+9	Perform calculation
EN	End of program

CR (Circle)

[Motion]

DESCRIPTION:

The CR command specifies a 2-dimensional arc segment of radius, r, starting at angle, θ, and traversing over angle Δθ. A positive Δθ denotes counter clockwise traverse, negative Δθ denotes clockwise. The VE (Vector Sequence End) command must be used to denote the end of the motion sequence after all CR and VP (Vector Position) segments are specified. TheBG (Begin) command is used to start the motion sequence. All parameters, r, θ, Δθ, must be specified. Radius units are in quadrature counts. θ and Δθ have units of degrees. The parameter n is optional and describes the vector speed that is attached to the motion segment.

ARGUMENTS: CR r,θ,Δθ<n>o where

r is an unsigned real number in the range 10 to 6000000 decimal (radius)

θ a signed number in the range 0 to +/-32000 decimal (starting angle in degrees)

Δθ is a signed real number in the range 0.0001 to +/-32000 decimal (angle in degrees)

n specifies a vector speed to be taken into effect at the execution of the vector segment, n is an unsigned even integer between 0 and 12,000,000 for servo motor operation and between 0 and 3,000,000 for stepper motors.

o specifies a vector speed to be achieved at the end of the vector segment, o is an unsigned even integer between 0 and 8,000,000.

Note: The product r * Δθ must be limited to +/-4.5 * 10^8

USAGE:

While Moving	Yes	Default Value	---
In a Program	Yes	Default Format	---
Command Line	Yes		
Virtual Axis	NO		

RELATED COMMANDS:

VP (Vector Position)

VS (Vector Speed)

VD (Vector Deceleration)

VA (Vector Acceleration)

VM (Coordinated Motion Mode)

VE (Vector Sequence End)

BG (Begin)

EXAMPLES:

VMAB	Specify vector motion in the A and B plane
VS 10000	Specify vector speed
CR 1000,0,360	Generate circle with radius of 1000 counts, start at 0 degrees and complete one circle in counterclockwise direction.
CR 1000,0,360<40000	Generate circle with radius of 1000 counts, start at 0 degrees and complete
VE	End Sequence
BGS	Start motion

CS (Clear Sequence)

[General]

DESCRIPTION:

The CS command will remove VP or LI commands stored in a motion sequence for the S or T coordinate systems. Please note that after a sequence has been run, the CS command is not necessary to enter a new sequence. This command is useful if you have incorrectly specified VP (Vector Position), CR (Circle) or LI (Linear Interpolation Distance) commands.

When used as an operand, _CS returns the number of the segment in the sequence, starting at zero. The instruction _CS is valid in LM (Linear Interpolation Mode), VM (Coordinated Motion Mode) and CM (Contour Mode).

ARGUMENTS: CSS or CST where

S and/or T can be used to clear the sequence buffer for the “S” or “T” coordinate system.

USAGE:

While Moving	No	Default Value	---
In a Program	Yes	Default Format	---
Command Line	Yes		

OPERAND USAGE:

_CSn contains the segment number in the sequence specified by n, S or T. The operand is valid in the Linear mode, LM, Vector mode, VM

RELATED COMMANDS:

CR (Circle)

LI (Linear Interpolation Distance)

LM (Linear Interpolation Mode)

VM (Coordinated Motion Mode)

VP (Vector Position)

EXAMPLES:

#CLEAR	Label
CAT	Specify the T coordinate system vector points
VP 1000,2000	Vector position
VP 4000,8000	Vector position
CST	Clear vectors specified in T coordinate system
CAS	Specify the T coordinate system vector points
VP 1000,5000	New vector
VP 8000,9000	New vector
CSS	Clear vectors specified in S coordinate system

CW (Copyright)

[General]

DESCRIPTION:

The CW command has a dual usage. The CW command will return the copyright information when the argument, n is 0. Otherwise, the CW command is used as a communications enhancement. When CW = 1, the communication enhancement sets the MSB of unsolicited, returned ASCII characters to 1. Unsolicited ASCII characters are those characters which are returned from the controller without being directly queried from an external source. This is the case when a program has a command that requires the controller to return a value or string. The benefit of this is that two-way unsolicited messages can be filtered by an external source to retrieve answers to strings that were sent by the external source. YTERM uses this feature to separate the incoming data into two buffers.

ARGUMENTS: CW n where

n is a number, either 0,1 or 2:

0	Causes the controller to return the copyright information
1	Causes the controller to set the MSB of unsolicited returned characters to 1
2	Causes the controller to not set the MSB of unsolicited characters.

USAGE:

While Moving	Yes*	Default Value	2
In a Program	Yes	Default Format	n/a
Command Line	Yes		
Can be Interrogated	No		
Used as an Operand	Yes		

OPERAND USAGE:

_CW contains the value of the data adjustment bit. 1 =on, 2 = off

***Note: The CW command can cause garbled characters to be returned by the controller. The default state of the controller is to disable the CW command, however, the Yaskawa Y-Term software will enable the CW command for internal usage. If the controller is reset while the Yaskawa software is running, the CW command could be reset to the default value which would create difficulty for the software. It may be necessary to re-enable the CW command. The CW command value is stored in EEPROM.**

DA (De-allocate Variables)

[General]

DESCRIPTION:

The DA command frees array and/or variable memory space. With this command, more than one array or variable can be specified for memory de-allocation. Different arrays and variables are separated by comma when specified in one command. The * argument de-allocates all variables, and *[0] de-allocates all arrays.

ARGUMENTS: DA c[0],d,etc. where

c[0] - Defined array name

d - Defined variable name

* - De-allocates all the variables

*[0] - De-allocates all the arrays

DA? Returns the number of arrays available on the controller.

USAGE:

While Moving	Yes	Default Value	---
In a Program	Yes	Default Format	---
Command Line	Yes		
Can be Interrogated	Yes		
Used as an Operand	Yes		

OPERAND USAGE:

_DA contains the total number of arrays available. For example, before any arrays have been defined, the operand _DA is 30. If one array is defined, the operand _DA will return 29.

RELATED COMMANDS:

DM (Dimension Array)

EXAMPLES:

'Cars' and 'Salesmen' are arrays and 'Total' is a variable.

Cars[400],Salesmen[50]	Dimension 2 arrays
Total=70	Assign 70 to the variable Total
DA Cars[0],Salesmen[0],Total	De-allocate the 2 arrays & variables
DA*[0]	De-allocate all arrays
DA *,*[0]	De-allocate all variables and all arrays

***Note: Since this command de-allocates the spaces and compacts the array spaces in the memory, it is possible that execution of this command may take longer time than 2 ms.**

DC (Deceleration)

[Motion]

DESCRIPTION:

The Deceleration command (DC) sets the linear deceleration rate for independent moves such as PR (Position Relative), PA (Position Absolute) and JG (Jog) moves. The parameters will be rounded down to the nearest factor of 1024 and have units of counts per second squared.

ARGUMENTS: DC x, y, z, w or DCX=x or DC a, b, c, d where

x, y z, w, or a, b, c, d are unsigned integers

USAGE:

While Moving	Yes*	Minimum Value	1024
In a Program	Yes	Maximum Value	67107840
Command Line	Yes	Default Value	256000
Can be Interrogated	Yes	Default Format	8.0
Used as an Operand	Yes		
Virtual Axis	Yes		

OPERAND USAGE:

_DCn contains the deceleration rate in counts/sec^2 where n is an axis letter.

RELATED COMMANDS:

AC (Acceleration)

PR (Position Relative)

SP (Speed)

JG (Jog)

BG (Begin)

IT (Independent Time Constant)

EXAMPLES:

PR 10000	Specify relative position
AC 2000000	Specify acceleration rate
DC 1000000	Specify deceleration rate
SP 5000	Specify slew speed
BG	Begin motion

***Note The DC command may be changed during the move in JG move, but not in PR or PA move. For controlled deceleration in abort conditions, use the ST command. The deceleration rate can be changed after ST.**

DE (Dual (Auxiliary) Encoder)

[Motion]

DESCRIPTION:

The DE command defines the position of the auxiliary encoder.

ARGUMENTS: DE x, y, z, w or DEX=x or DE a, b, c, d where

x, y z, w, or a, b, c, d are signed integers

USAGE:

While Moving	Yes	Minimum Value	-2147483647
In a Program	Yes	Maximum Value	2147483648
Command Line	Yes	Default Value	n/a
Can be Interrogated	Yes	Default Format	Position Format
Used as an Operand	Yes		
Virtual Axis	No		

OPERAND USAGE:

_DEn returns the current position of the specified auxiliary encoder where n is an axis letter.

EXAMPLES:

:DE 0	Set the auxiliary encoder position to 0
:MG_DEX	Return auxiliary encoder positions
:DUALX=_DE	Assign auxiliary encoder position of X-axis to the variable DUALX

HINT: ***Dual encoders are useful when you need an encoder on the motor and on the load. The encoder on the load is typically the auxiliary encoder and is used to verify the true load position. Any error in load position is used to correct the motor position.***

DL (Download)

[General]

DESCRIPTION:

The DL command prepares a controller to accept a data file from the host computer. Instructions in the file will be accepted as a data stream without line numbers. The file is terminated using <control> Z, <control> Q, <control> D, or \.

If no parameter is specified, downloading a data file will clear any programs in the SMC–4000 RAM. The data is entered beginning at line 0. If there are too many lines or too many characters per line, the SMC–4000 will return a "?". To download a program after a label, specify the label name following DL. The # argument may be used with DL to append a file at the end of the SMC–4000 program in RAM.

ARGUMENTS: DL n where

n = no argument Downloads program beginning at line 0 and erases programs in RAM.

n = #Label Begins download at line following #Label where label may be any valid program label.

n = # Begins download at end of program in RAM.

USAGE:

While Moving	Yes	Default Value	---
In a Program	No	Default Format	---
Command Line	Yes		
Can be Interrogated	No		
Used as an Operand	Yes		

OPERAND USAGE:

When used as an operand, _DL gives the number of available labels. The total number of labels is 510.

RELATED COMMANDS:

UL (Upload)

EXAMPLES (from the terminal):

DL;	Begin download (no colon returned)
#A;PR 4000;BG	Data
AM;MG DONE	Data
EN	Data
<control> Z	End download (colon returned)

DM (Dimension Array)

[General]

DESCRIPTION:

The DM command defines a single dimensional array with a name and total elements. The first element of the defined array starts with element number 0 and the last element is at n-1.

Note: If an array is already defined and a new size is required, first use the DA command to de-allocate the array.

ARGUMENTS: DM c[n] where

c is a name of up to eight alphanumeric characters, starting with an uppercase alphabetic character.

n is the number of entries from 1 to 15000.

USAGE:

While Moving	Yes	Default Value	---
In a Program	Yes	Default Format	---
Command Line	Yes		
Can be Interrogated	Yes		
Used as an Operand	Yes		

OPERAND USAGE:

_DM contains the available array space. For example, before any arrays have been defined, the operand _DM will return 15000. If an array of 100 elements is defined, the operand _DM will return 14900.

RELATED COMMANDS:

DA (De-allocate Variables)

EXAMPLES:

DM Pets[5],Dogs[2],Cats[3]	Define dimension of arrays, pets with 5 elements; Dogs with 2 elements; Cats with 3 elements
DM Tests[1000]	Define dimension of array called Tests with 1000 elements

DP (Define Position)

[Setting]

DESCRIPTION:

The DP command sets the current motor position and current command positions to a user specified value. The units are in quadrature counts. This command will set both the TP (Tell Position) and RP (Reference Position) values.

ARGUMENTS: DP x, y, z, w or DPX=x or DP a, b, c, d where

x, y z, w, or a, b, c, d are signed integers

USAGE:

While Moving	No	Minimum Value	-2147483648
In a Program	Yes	Maximum Value	+2147483647
Command Line	Yes	Default Value	n/a
Can be Interrogated	Yes	Default Format	Position Format
Used as an Operand	Yes		
Virtual Axis	Yes		

OPERAND USAGE:

_DPn reports the current position where n is an axis letter.

EXAMPLES:

:DP 0	Sets the current position of the X axis to 0
:DP -50000	Sets the current position to -50000.
:MG_DPX	
-0050000	Returns the motor position

DT (Delta Time)

[Motion]

DESCRIPTION:

The DT command sets the time interval for Contouring Mode. Sending the DT command once will set the time interval for all following contour data until a new DT command is sent. 2^n samples is the time interval. Sending DT0 followed by CD0 command terminates the Contour Mode.

ARGUMENTS: DT n where

n is an integer. 0 terminates the Contour Mode.

n=1 thru 8 specifies the time interval of 2^n samples. By default the sample period is 1 msec (set by TM command); with n=1, the time interval would be 2 msec.

USAGE:

While Moving	Yes	Minimum Value	0
In a Program	Yes	Maximum Value	8
Command Line	Yes	Default Value	0
Can be Interrogated	Yes	Default Format	1.0
Used as an Operand	Yes		

OPERAND USAGE:

_DT contains the value for the time interval for Contour Mode

RELATED COMMANDS:

CM (Contour Mode)

CD (Contour Data)

WC (Wait for Contour)

EXAMPLES:

DT 4	Specifies time interval to be 16 msec
DT 7	Specifies time interval to be 128 msec
#CONTOUR	Begin
CM	Enter Contour Mode
DT 4	Set time interval
CD 1000	Specify data
WC	Wait for contour
CD 2000	New data
WC	Wait
DT0	Stop contour
CD0	Exit Contour Mode
EN	End

DV (Dual Velocity (Dual Loop))

[Configuration]

DESCRIPTION:

The DV function changes the operation of the PID servo loop. It causes the KD (Derivative Constant) term to operate on the dual encoder instead of the main encoder. This results in improved stability in the cases where there is a backlash between the motor and the main encoder, and where the dual encoder is mounted on the motor. See the example section.

ARGUMENTS: DV x, y, z, w or DVX=x or DV a, b, c, d where

x, y z, w, or a, b, c, d are unsigned integers

0 disables the function. 1 enables the dual loop.

USAGE:

While Moving	Yes	Minimum Value	0
In a Program	Yes	Maximum Value	1
Command Line	Yes	Default Value	0
Can be Interrogated	Yes	Default Format	1
Used as an Operand	Yes		
Virtual Axis	No		

OPERAND USAGE:

_DVn contains the state of dual velocity mode where n is an axis letter and 0 = disabled, 1 = enabled.

RELATED COMMANDS:

KD (Derivative Constant)

FV (Velocity Feedforward)

EXAMPLES:

DV 1	Enables dual loop PID
DV 0	Disables DV

HINT: The DV command is useful in backlash and resonance compensation.

EA (ECAM Master)

[Setting]

DESCRIPTION:

The EA command selects the master axis for the electronic cam mode.

ARGUMENTS: EA x where

x: X Y Z W N

USAGE:

While Moving	Yes	Default Value	n/a
In a Program	Yes	Default Format	n/a
Command Line	Yes		
Virtual Axis	Yes		

RELATED COMMANDS:

EB (ECAM Enable)

EC (ECAM Counter)

EG (ECAM Engage)

EM (ECAM Cycle)

EP (ECam Table Intervals and Start Point)

EQ (ECam Quit (Disengage))

ET (ECam Table)

EXAMPLES:

EAN	Select virtual axis as the master for ECAM

EB (ECAM Enable)

[Setting]

DESCRIPTION:

The EB function enables or disables the cam mode. In this mode, the master axis is modularized within the cycle. This command does not initiate camming but it readies the controller for cam mode.

ARGUMENTS: EB n where

n = 1 starts cam mode and n = 0 stops cam mode.

USAGE:

While Moving	Yes	Default Value	0
In a Program	Yes	Default Format	n/a
Command Line	Yes		
Can be Interrogated	Yes		
Used as an Operand	Yes		

OPERAND USAGE:

_EB contains the state of Ecam mode. 0 = disabled, 1 = enabled

RELATED COMMANDS:

EM (ECAM Cycle)

EP (ECam Table Intervals and Start Point)

EXAMPLES:

EB1	Starts ECAM mode
EB0	Stops ECAM mode
B = _EB	Return status of cam mode

***Note See the example section for more detailed cam examples.**

EC (ECAM Counter)

[Setting]

DESCRIPTION:

The EC function sets the index into the ECAM table. This command is only useful when entering ECAM table values without index values when only certain sections of the cam data must be changed. See the ET (ECam Table) command.

ARGUMENTS: EC n where

n is an integer between 0 and 1024.

USAGE:

While Moving	Yes	Default Value	---
In a Program	Yes	Default Format	---
Command Line	Yes		
Can be Interrogated?	Yes		
Used as Operand?	Yes		

OPERAND USAGE:

_EC contains the current value of the index into the ECAM table.

RELATED COMMANDS:

EA (ECAM Master)

EB (ECAM Enable)

EG (ECAM Engage)

EP (ECam Table Intervals and Start Point)

ET (ECam Table)

EM (ECAM Cycle)

EQ (ECam Quit (Disengage))

EXAMPLES:

EC0	Set ECAM index to 0
ET 200,400	Set first ECAM table entries to 200,400
ET 400,800	Set second ECAM table entries to 400,800

ED (Edit Mode)

[General]

DESCRIPTION:

The ED command puts the controller into the Edit subsystem. In the Edit subsystem, programs can be created, changed or destroyed. The commands in the Edit subsystem are:

<cntrl>D	Deletes a line
<cntrl>I	Inserts a line before the current one
<cntrl>P	Displays the previous line
<cntrl>Q	Exits the Edit subsystem
<return>	Saves a line

Because the download time for a complete program is usually very short, Yaskawa recommends all editing be performed by Yaskawa's YTerm software. This command is primarily documented for it's usefulness when a command error occurs. _ED indicates the line that had the error.

ARGUMENTS: ED n where

n specifies the line number to begin editing. The default line number is the last line of program space with commands.

USAGE:

While Moving	No	Default Value	n/a
In a Program	No	Default Format	n/a
Command Line	Yes		
Can be Interrogated	No		
Used as an Operand	Yes		

OPERAND USAGE:

_ED contains the line number of the last line to have an error. Very useful in tracing problems in the field.
_ED1 contains the thread that had an error.
_ED2 contains the instruction number that caused the error.
_ED3 contains the instruction after the line that caused the error.
XQ _ED1 _ED2, restart a thread

EXAMPLES:

000 #START	
001 PR 2000	
002 BG	
003 SLKJ	Bad line
004 EN	
005 #CMDERR	Routine which occurs upon a command error
006 V=_ED	
007 MG "An error has occurred" {n}	
008 MG "In line", V{F3.0}	
009 ST	
010 ZS0	
011 EN	

HINT: Remember to quit the Edit Mode prior to executing or listing a program.

EG (ECAM Engage)

[Motion]

DESCRIPTION:

The EG command engages an ECAM operation at a specified position of the master encoder. If a value is specified outside of the master's range, the slave will engage immediately. Once a slave motor is engaged, its position is redefined to fit within the cycle.

ARGUMENTS: EG x, y, z, w or EG n =where

n is the master position at which the slave axis must be engaged.

USAGE:

While Moving	Yes	Minimum value	-2147483648
In a Program	Yes	Maximum value	2147483647
Command Line	Yes	Default Value	n/a
Can be Interrogated	No	Default Format	n/a
Used as an Operand	Yes		
Virtual Axis	No		

OPERAND USAGE:

_EGn contains ECAM status where n is an axis letter. 0 = axis is not engaged, 1 = axis is engaged.

RELATED COMMANDS:

EA (ECAM Master)

EB (ECAM Enable)

EC (ECAM Counter)

EM (ECAM Cycle)

EP (ECam Table Intervals and Start Point)

EQ (ECam Quit (Disengage))

ET (ECam Table)

EXAMPLES:

EG 700	Engages slave at master position 700.
B = _EGB	Return the status of the axis, 1 if engaged

***Note This command is not a trippoint. This command will not hold the execution of the program flow. If the execution needs to be held until master position is reached, use MF or MR command.**

ELSE

[Program Flow]

DESCRIPTION:

The ELSE command is an optional part of an IF conditional statement. The ELSE command must occur after an IF command and it has no arguments. It allows for the execution of a command only when the argument of the IF command evaluates False. If the argument of the IF command evaluates false, the controller will skip commands until the ELSE command. If the argument for the IF command evaluates true, the controller will execute the commands between the IF and ELSE command.

ARGUMENTS: None

USAGE:

While Moving	Yes	Default Value	---
In a Program	Yes	Default Format	---
Command Line	No		

RELATED COMMANDS:

ENDIF End of IF conditional Statement

EXAMPLES:

IF (@IN[1]=0)	IF conditional statement based on input 1
IF (@IN[2]=0)	2nd IF conditional statement executed if 1st IF conditional true
MG "INPUT 1 AND INPUT 2 ARE ACTIVE"	Message to be executed if 2nd IF conditional is true
ELSE	ELSE command for 2nd IF conditional statement
MG "ONLY INPUT 1 IS ACTIVE	Message to be executed if 2nd IF conditional is false
ENDIF	End of 2nd conditional statement
ELSE	ELSE command for 1st IF conditional statement
MG"ONLY INPUT 2 IS ACTIVE"	Message to be executed if 1st IF conditional statement
ENDIF	End of 1st conditional statement

EM (ECAM Cycle)

[Setting]

DESCRIPTION:

The EM command is part of the ECAM mode. It is used to define the change in position over one complete cycle of the slave. If a slave will return to its original position at the end of the cycle, the change is zero. If the change is negative, specify the absolute value.

ARGUMENTS: EM x, y, z, w or EMX = w where

x, y, z, w is the net change in the axis during camming.

USAGE:

While Moving	Yes	Minimum n parameter	-2147483648
In a Program	Yes	Maximum n parameter	2147483647
Command Line	Yes	Default Value	0
Can be Interrogated	No	Default Format	---
Used as an Operand	Yes		
Virtual Axis	Yes		

OPERAND USAGE:

_EMn contains the cam cycle of the slave where n is an axis letter.

RELATED COMMANDS:

EB (ECAM Enable)

EP (ECam Table Intervals and Start Point)

ET (ECam Table)

EXAMPLES:

EM 2000	Define the net change in the slave to be 2000.
V = _EM	Return slave's cam cycle distance
EMN = 10000	Define virtual master cycle

EN (End)

[Program Flow]

DESCRIPTION:

The EN command is used to designate the end of a program or subroutine. If a subroutine was called by the JS (Jump to Subroutine) command, the EN command ends the subroutine and returns program flow to the command just after JS.

ARGUMENTS: None

***Note Use the RE command to return from the interrupt handling subroutines #LIMSWI and #POSERR. Use the RI command to return from the #ININT subroutine.**

USAGE:

While Moving	Yes	Default Value	n/a
In a Program	Yes	Default Format	n/a
Command Line	No		
Can be Interrogated	No		
Used as an Operand	No		

RELATED COMMANDS:

RE (Return from Error)

RI (Return from Interrupt)

EXAMPLES:

#A	Program A
PR 500	Move X axis forward 500 counts
BGX	Move X axis forward 1000 counts
AMX	Pause the program until the X axis completes the motion
PR 1000	Set another Position Relative move
BGX	Begin motion
EN	End of Program

***Note Instead of EN, use the RE command to end the error subroutine and limit subroutine. Use the RI command to end the input interrupt) subroutine.**

ENDIF

[Program Flow]

DESCRIPTION:

The ENDIF command is used to designate the end of an IF conditional statement. An IF conditional statement is formed by the combination of an IF and ENDIF command. An ENDIF command must always be executed for every IF command that has been executed. It is recommended that the user not include jump commands inside IF conditional statements since this causes re-direction of command execution. In this case, the command interpreter may not execute an ENDIF command.

ARGUMENTS: ENDIF

USAGE:

While Moving	Yes	Default Value	n/a
In a Program	Yes	Default Format	n/a
Command Line	No		

RELATED COMMANDS:

ELSE Optional command to be used only after IF command

JP (Jump to Program Location)

JS (Jump to Subroutine)

EXAMPLES:

IF (@IN[1]=0)	IF conditional statement based on input 1
"MG " INPUT 1 IS ACTIVE	Message to be executed if “IF” conditional is true
ENDIF	End of conditional statement

EO (Echo)

[Setting]

DESCRIPTION:

The EO command turns the echo on or off. If the echo is off, characters input to the serial port or Ethernet will not be echoed back.

Note: Commands sent over Ethernet are not echoed.

ARGUMENTS: EO n where

n=0 or 1. 0 turns echo off, 1 turns echo on.

USAGE:

While Moving	Yes	Default Value	1
In a Program	Yes	Default Format	1
Command Line	Yes		
Can be Interrogated	No		
Used as an Operand	No		

EXAMPLES:

EO 0	Turns echo off
EO 1	Turns echo on

EP (ECam Table Intervals and Start Point)

[Setting]

DESCRIPTION:

The EP command defines the ECAM table intervals and offset. The offset is the master position where the first ECAM table entry will synchronize. The interval is the difference of the master position between any two consecutive table entries. This command effectively defines the size of the ECAM table. The parameter m is the interval and n is the starting point. Up to 1024 points may be specified using the ET (ECam Table) command.

ARGUMENTS: EP m,n where

m, n are signed integers

USAGE:

While Moving	Yes	Minimum m value	1
In a Program	Yes	Minimum m value	32767
Command Line	Yes	Minimum n value	-2147483648
Can be Interrogated	Yes	Minimum n value	2147483647
Used as an Operand	Yes (m only)		

OPERAND USAGE:

_EP contains the value of the interval m.

RELATED COMMANDS:

EB (ECAM Enable)

EG (ECAM Engage)

EM (ECAM Cycle)

EQ (ECam Quit (Disengage))

ET (ECam Table)

EXAMPLES:

EP 20,100	Sets the cam master points to 100,120,140 . . .
D = _EP	Returns interval (m)

EQ (ECam Quit (Disengage))

[Motion]

DESCRIPTION:

The EQ command disengages an electronic cam slave axis at the specified master position. If a value is specified outside of the master's range, the slave will disengage immediately.

ARGUMENTS: EQ n where

n is the master position at which the axis is to be disengaged.

USAGE:

While Moving	Yes	Minimum value	-2147483647
In a Program	Yes	Maximum value	2147483648
Command Line	Yes	Default Value	--
Can be Interrogated	Yes	Default Format	--
Used as an Operand	Yes		
Virtual Axis	No		

OPERAND USAGE:

_EQn contains 1 if engage command is issued and slave is waiting to engage, 2 if disengage command is issued and slave is waiting to disengage, and 0 if ECAM engaged or disengaged.

RELATED COMMANDS:

EB (ECAM Enable)

EG (ECAM Engage)

EM (ECAM Cycle)

EP (ECam Table Intervals and Start Point)

ET (ECam Table)

EXAMPLES:

EQ 300	Disengages the motor at master position 300.

***Note This command is not a trippoint. This command will not hold the execution of the program flow. If the execution needs to be held until master position is reached, use MF or MR command.**

ER (Error Limit)

[Setting]

DESCRIPTION:

The ER command sets the magnitude of the position error that will trigger an error condition. When the limit is exceeded, the Error LED will illuminate. If the Off-On-Error (OE1) command is active, the amplifier will be disabled. The units of ER are quadrature counts. An ER value of 0 will disable the error function, meaning that a #POSERR in the program will not execute, the red alarm LED will not illuminate for excessive following error, and the motor will not be disabled if OE (Off On Error) is set.

ARGUMENTS: ***ER x, y, z, w or ERX=x or ER a, b, c, d*** ***where***

x, y z, w, or a, b, c, d are unsigned integers

USAGE:

While Moving	Yes	Minimum Value	0
In a Program	Yes	Maximum Value	32767
Command Line	Yes	Default Value	16384
Can be Interrogated	Yes	Default Format	Position Format
Used as an Operand	Yes		
Virtual Axis	No		

OPERAND USAGE:

_ERn contains the value of the ERror limit where n is an axis letter.

RELATED COMMANDS:

OE (Off On Error)

#POSERR Automatic Error Subroutine

EXAMPLES:

ER 200	Set the error limit to 200
MG_ERX	Return value
00200	
V1=_ER	Assigns V1 value of ER
V1=	Returns V1
00200	

HINT: ***The error limit specified by ER should be high enough as not to be reached during normal operation. Examples of exceeding the error limit would be a mechanical jam, or a fault in a system component such as encoder or amplifier.***

ES (Ellipse Scale)

DESCRIPTION:

The ES command divides the resolution of one of the axes in a vector mode (VM). This function allows for the generation of circular motion when encoder resolutions differ. It also allows for the generation of an ellipse instead of a circle.

The command has two parameters, m and n. The arguments, m and n, apply to the axes designated by the VM (Coordinated Motion Mode) command. When m>n, the resolution of the first axis, x, will be divided by the ratio m/n. When m<n, the resolution of the second axis, y, will be divided by n/m. The resolution change applies for the purpose of generating the VP (Vector Position) and CR (Circle) commands, effectively changing the axis with the higher resolution to match the coarser resolution.

The ES command will apply to the selected coordinate system, S or T. To select the coordinate system, use the command CAS or CAT.

ARGUMENTS: ***ES m,n*** ***where***

m and n are positive integers in the range between 1 and 65,535.

USAGE:

While Moving	Yes	Default Value	1,1
In a Program	Yes	Default Format	
Command Line	Yes		

RELATED COMMANDS:

VM (Coordinated Motion Mode)

CR (Circle)

VP (Vector Position)

EXAMPLES:

VMAB;ES3,4	Divide B resolution by 4/3
VMCA;ES2,3	Divide A resolution by 3/2
VMAC;ES3,2	Divide A resolution by 3/2

ET (ECam Table)

[Setting]

DESCRIPTION:

The ET command sets the ECAM table entries for the slave axis. The values of the master are not required. The slave entry (n) is the position of the slave when the master is at the point (n ∗ i) + o, where i is the interval and o is the offset as determined by the EP (ECam Table Intervals and Start Point) command.

ARGUMENTS: ET [n] = m or ET[n] = a, b, c, d where

n is an integer between 0 and 1024.

m is an integer between -2147483648 and 2147483647.

USAGE:

While Moving	Yes	Minimum n Value	0
In a Program	Yes	Maximum n Value	1024
Command Line	Yes	Default n Value	n/a
Can be Interrogated	No	Minimum m Value	-2147483648
Used as an Operand	No	Maximum m Value	2147483647
		Default m Value	n/a
Virtual Axis	No		

RELATED COMMANDS:

EB (ECAM Enable)

EG (ECAM Engage)

EM (ECAM Cycle)

EP (ECam Table Intervals and Start Point)

EQ (ECam Quit (Disengage))

EXAMPLES:

ET [7] = 1000	Specifies the position of the slave that must be synchronized with the eighth increment of the master.
ET[x] = XPOS, YPOS, ZPOS, WPOS	Specifies multiple slave axes synchronized to increment *[x]* of the master.

FA (Acceleration Feedforward)

[Setting]

DESCRIPTION:

The FA command sets the acceleration feedforward coefficient, or returns the previously set value. This coefficient, when scaled by the acceleration, adds a torque bias voltage during the acceleration phase and subtracts the bias during the deceleration phase of a motion.

Acceleration Feedforward Bias = FA ∗ AC ∗ 1.5 ∗ 10^{-7}

Deceleration Feedforward Bias = FA ∗ DC ∗ 1.5 ∗ 10^{-7}

The Feedforward Bias product is limited to 10 Volts. FA will only be operational during independent moves, not gearing, camming or interpolation.

ARGUMENTS: FA x, y, z, w or FAX=x or FA a, b, c, d where

x, y z, w, or a, b, c, d are unsigned numbers

FA has a resolution of 0.25

USAGE:

While Moving	Yes	Minimum Value	0
In a Program	Yes	Maximum Value	8191
Command Line	Yes	Default Value	0
Can be Interrogated	Yes	Default Format	4.0
Used as an Operand	Yes		
Virtual Axis	No		

OPERAND USAGE:

_FAn contains the value of the feedforward acceleration coefficient where n is an axis letter.

RELATED COMMANDS:

FV (Velocity Feedforward)

EXAMPLES:

AC 500000	Set acceleration
FA 10	Set feedforward coefficient to 10
MG_FAX	The effective bias will be 0.75V (10 ∗ 500000 ∗ 1.5 ∗ 10^{-7})
010	Return value

***Note If the feedforward coefficient is changed during a move, then the change will not take effect until the next move.**

FE (Find Edge)

[Motion]

DESCRIPTION:

The FE command moves a motor until a transition is seen on the homing input for the associated axis. The direction of motion depends on the initial state of the homing input (use the CN (Configure Limit Switches) command to configure the polarity of the home input). Once the transition is detected, the motor decelerates to a stop.

This command is useful for creating your own homing sequences**. See the example section.**

ARGUMENTS: *FE XYZW or ABCD*

USAGE:

While Moving	No	Default Value	---
In a Program	Yes	Default Format	---
Command Line	Yes		
Can be Interrogated	No		
Used as an Operand	No		
Virtual Axis	No		

RELATED COMMANDS:

FI (Find Index)

HM (Home)

BG (Begin)

AC (Acceleration)

DC (Deceleration)

SP (Speed)

EXAMPLES:

FE XY	Set find edge mode
BG XY	Begin

HINT: ***Find Edge only searches for a change in state on the Home Input. Use FI (Find Index) to search for the encoder "C" channel. Remember to specify BG (Begin) after each of these commands. Use HM (Home) to search for both the Home input and the Index.***

FI (Find Index)

[Motion]

DESCRIPTION:

The FI and BG (Begin) combination moves the motor until an encoder index pulse, or “C” channel, is detected. The controller looks for a transition from low to high. When the transition is detected, motion stops and the position is defined as zero. To improve accuracy, the speed during the search should be specified as 1000 counts/s or less. The FI command is useful in custom homing sequences. The direction of motion is specified by the sign of the JG (Jog) command.

ARGUMENTS: FI XYZW or ABCD

USAGE:

While Moving	No	Default Value	---
In a Program	Yes	Default Format	---
Command Line	Yes		
Can be Interrogated	No		
Used as an Operand	No		
Virtual Axis	No		

RELATED COMMANDS:

FE (Find Edge)

HM (Home)

BG (Begin)

AC (Acceleration)

DC (Deceleration)

JG (Jog)

EXAMPLES:

#HOME	Home Routine
JG 500	Set speed and forward direction
FI	Find index
BG	Begin motion
AM	After motion
MG "FOUND INDEX"	
EN	

HINT: ***Find Index only searches for a change in state on the Index. Use FE (Find Edge) to search for the Home input. Use HM (Home) to search for both the Home input and the Index. Remember to specify BG (Begin) after each of these commands.***

FL (Forward Limit)

[Setting]

DESCRIPTION:

The FL command sets the forward software position limit. If this limit is exceeded during commanded motion, the motor will decelerate to a stop. Forward motion beyond this limit is not permitted. The forward limit is activated at position n + 1. The forward limit is disabled at position 2147483647. The units are in counts.

When the reverse software limit is activated, the automatic subroutine #LIMSWI will be executed if it is included in the program and the program is executing. See section on Automatic Subroutines.

ARGUMENTS: FL x, y, z, w or FLX=x or FL a, b, c, d where

x, y z, w, or a, b, c, d are unsigned numbers

USAGE:

While Moving	Yes	Minimum Value	-2147483648
In a Program	Yes	Maximum Value	+2147483647
Command Line	Yes	Default Value	2147483647
Can be Interrogated	Yes	Default Format	Position Format
Used as an Operand	Yes		
Virtual Axis	No		

OPERAND USAGE:

_FLn contains the value of the forward software limit where n is an axis letter.

RELATED COMMANDS:

BL (Backward Limit)

EXAMPLES:

#TEST	Test Program
AC 1000000	Acceleration Rate
DC 1000000	Deceleration Rate
FL 15000	Forward Limit
JG 5000	Jog Forward
BGX	Begin
AMX	After Motion
TPX	Tell Position
EN	End

@FRAC (Fraction)

[Function]

DESCRIPTION:

@FRAC returns only the fractional portion of a number or variable given in square brackets. Note that the @FRAC command is a function, which means that it does not follow the convention of the commands, and does not require the underscore when used as an operand.

ARGUMENTS: @FRAC [n] where

n is a number

USAGE:

While Moving	Yes	Minimum n value	-2147483647.9999
In a Program	Yes	Maximum n value	2147483647.9999
Not in a program	Yes	Default Value	---
Can be Interrogated	No	Default Format	---
Used as an Operand	Yes		

EXAMPLES:

#TEST	Program TEST
VAR1=123.456	Set variable
MG @FRAC[VAR1]	Display only the fractional portion of VAR1
VAR2=@FRAC[VAR1]+.5	Perform calculation
EN	End of program

FV (Velocity Feedforward)

[Setting]

DESCRIPTION:

The FV command sets the velocity feedforward coefficient, or returns the previously set value. This coefficient generates an output bias signal in proportion to the commanded velocity.

Velocity feedforward bias = 1.22 x 10^{-6} x FV x Velocity [in ct/s].

For example, if FV=10 and the velocity is 200,000 count/s, the velocity feedforward bias equals 2.44 volts.

ARGUMENTS: FV x, y, z, w or FVx=x or FV a, b, c, d where

x, y, z, w, or a, b, c, d are unsigned numbers

USAGE:

While Moving	Yes	Minimum Value	0
In a Program	Yes	Maximum Value	8192
Command Line	Yes	Default Value	0
Can be Interrogated	Yes	Default Format	n/a
Used as an Operand	Yes		
Virtual Axis	No		

OPERAND USAGE:

_FV contains the velocity feedforward coefficient where n is an axis letter.

RELATED COMMANDS:

FA (Acceleration Feedforward)

EXAMPLES:

FV 10	Set feedforward coefficients to 10
JG 30000	This speed produces 0.366 volts of torque offset (1.22 X 10^{-6} ∗ 10 ∗ 30000).
FV ?	Return the value
010	

GA (Master Axis for Gearing)

[Setting]

DESCRIPTION:

The GA command specifies the master axes for electronic gearing. Multiple masters for gearing may be specified. The masters may be the main encoder input, auxiliary encoder input, or the commanded position of any axis. The master may also be the commanded vector move in a coordinated motion of LM (Linear Interpolation Mode) or VM (Coordinated Motion Mode) type. When the master is a simple axis, it may move in any direction and the slave follows. When the master is a commanded vector move, the vector move is considered positive and the slave will move forward if the gear ratio is positive, and backward if the gear ratio is negative. The slave axes and ratios are specified with the GR (Gear Ratio) command and gearing is turned off by the command GR0.

ARGUMENTS: ***GA n,n,n,n,*** ***or*** ***GAA=n*** ***where***

n can be A,B,C,D or N. The value of x is used to set the specified main encoder axis as the gearing master and N represents the virtual axis. The slave axis is specified by the position of the argument. The first position of the argument corresponds to the 'A' axis, the second position corresponds to the 'B' axis, etc. A comma must be used in place of an argument if the corresponding axes will not be a slave.

n can be CA,CB,CC,CD. The value of x is used to set the commanded position of the specified axis as the gearing master.

n can be S or T. S and T are used to specify the vector motion of the coordinated system, S or T, as the gearing master.

n can be DA,DB,DC,DD. The value of n is used to set the specified auxiliary encoder axis as the gearing master.

USAGE:

While Moving	No	Default Value	---
In a Program	Yes	Default Format	---
Command Line	Yes		
Controller Usage	All controllers		
Virtual Axis	Yes		

RELATED COMMANDS:

GR (Gear Ratio)

GM (Gantry Mode)

EXAMPLES:

#GEAR	Gear program
GA ,A,T	Specify A axis as master for B and vector motion on T as master for C
GR ,.5,-2.5	Specify B and C ratios
JG 5000	Specify master jog speed
BGA	Begin motion
WT 10000	Wait 10000 msec
STA	Stop

HINT: ***Using the command position as the master axis is useful for gantry applications. Using the vector motion as master is useful in generating Helical motion.***

GM (Gantry Mode)

DESCRIPTION:

The GM command specifies the axes in which the gearing function is performed in the Gantry Mode. In this mode, the gearing will not be stopped by the ST (Stop) command or by limit switches. Only GR0 will stop the gearing in this mode.

ARGUMENTS: GM n,n,n,n or GMA=n where

n = 0 Disables gantry mode function

n = 1 Enables the gantry mode

USAGE:

While Moving	Yes	Default Value	0
In a Program	Yes	Default Format	1.0
Command Line	Yes		

OPERAND USAGE:

_GMn contains the state of gantry mode for the specified axis; 0 gantry mode disabled, 1 gantry mode enabled.

RELATED COMMANDS:

GR (Gear Ratio)

GA (Master Axis for Gearing)

EXAMPLES:

GM 1,1,1,1	Enable GM on all axes
GM 0	Disable GM on A-axis, other axes remain unchanged
GM ,,1,1	Enable GM on C-axis and D-axis, other axes remain unchanged
GM 1,0,1,0	Enable GM on A-axis and C-axis, disable GM on B and D-axis

HINT: The GM command is useful for driving a heavy load on both sides (Gantry Style).

GR (Gear Ratio)

[Motion]

DESCRIPTION:

GR specifies the Gear Ratio for the slave axis in electronic gearing mode. The master axis for the SMC–4000 is specified with the GA (Master Axis for Gearing) command. Gear ratio may range between +/-127.9999. The slave axis will be geared to the actual position of the master. The master can go in both directions. GR 0 disables gearing. If a limit switch is hit during gearing, then gearing is automatically disabled.

ARGUMENTS: GR n where

n is a signed number.

0 disables gearing

USAGE:

While Moving	Yes	Minimum Value	-127.9999
In a Program	Yes	Maximum Value	127.9999
Command Line	Yes	Default Value	0
Can be Interrogated	Yes	Default Format	3.4
Used as an Operand	Yes		
Virtual Axis	Yes		

OPERAND USAGE:

_GRn contains the value of the gear ratio where n is an axis letter.

EXAMPLES:

#GEAR	
GR .25	Specify gear ratio
EN	End program

Note: The gear ratio value entered has a resolution of 1/16. This means that a gear ratio of 1.35 will be stored as 1.349996. To check a given ratio, multiply the fractional portion by 65535 (0.4 × 65535 = 26214.4). Only the integer portion is stored, so the actual fraction is 26214/65535.

HM (Home)

[Motion]

DESCRIPTION:

The HM command performs a three-stage homing sequence.

The first stage is the motor moving at the user programmed speed until detecting a transition on the Home input. The direction for this first stage is determined by the initial state of the Home input. Once the Home input changes, the motor decelerates to a stop. The state of the Home input can be configured using the CN (Configure Limit Switches) command.

The second stage consists of the motor changing directions and slowly approaching the transition again. When the transition is detected, the motor is stopped instantaneously.

The third stage consists of the motor slowly moving forward until it detects an index pulse from the encoder. It stops at this point and defines it as position 0.

ARGUMENTS: HM XYZW or ABCD

USAGE:

While Moving	No	Default Value	---
In a Program	Yes	Default Format	---
Command Line	Yes		
Can be Interrogated	No		
Used as an Operand	Yes		
Virtual Axis	No		

OPERAND USAGE:

_HMn contains the state of the Home input. Regardless of the limit switch polarity, where n is an axis letter, 0 always means the home input is active, 1 means inactive.

RELATED COMMANDS:

CN (Configure Limit Switches)

FI (Find Index)

FE (Find Edge)

EXAMPLES:

HM	Set Homing Mode
BG	Begin Homing

HINT: You can customize homing sequence by using the FE (Find Edge) and FI (Find Index) commands.

HS (Handle Switch)

[Configuration]

DESCRIPTION:

The HS command is used to switch the handle assignments between two handles. Handles are assigned explicitly with the IH (Internet Handle) command, or automatically if a remote device initiates the connection. Should those assignments need modifications, the HS command allows the handles to be reassigned. This is very useful if using the MB (Modbus), MG (Message), or SA (Send Command) commands, which expect a certain device to be present on a specific handle.

ARGUMENTS: HS a=b where

a = the first handle of the switch (A - H)

b = the second handle of the switch (A - H)

USAGE:

While Moving	Yes	Default Value	n/a
In a Program	Yes	Default Format	n/a
Command Line	Yes		
Can be Interrogated	No		
Used as an Operand	No		

RELATED COMMANDS:

IH (Internet Handle)

EXAMPLES:

HSC=D	Connection for handle C is assigned to handle D. Connection for handle D is assigned to handle C.

This example demonstrates a master controller, M2, searching for other controllers on each handle and forcing them to communicate at a specific handle. This is necessary when using the @IN (Input), SB (Set Bit), CB (Clear Bit), SA (Send Command) or MB (Modbus) commands, because they reference specific handles.

The Jump condition logic below ((_IHA0+4)<>_IA)|(_IHA2<>2) is basically running the SH (Servo Here) command on the given line if the IP address is not equal to the controller own IP address + 4 and if the connection type is not TCP/IP.

```
MG "M2 SEARCHING FOR M1 AND M3"
#FIND_M1
  JP#FIND_M1+2,((_IHA0+4)<>_IA)|(_IHA2<>2); MG"M1 ALREADY ON A"; JP #FIND_M3
  JP#FIND_M1+3,((_IHB0+4)<>_IA)|(_IHB2<>2); HSB=A; MG"M1 FROM B TO A"; JP #FIND_M3
  JP#FIND_M1+4,((_IHC0+4)<>_IA)|(_IHC2<>2); HSC=A; MG"M1 FROM C TO A"; JP #FIND_M3
  JP#FIND_M1+5,((_IHD0+4)<>_IA)|(_IHD2<>2); HSD=A; MG"M1 FROM D TO A"; JP #FIND_M3
  JP#FIND_M1+6,((_IHE0+4)<>_IA)|(_IHE2<>2); HSE=A; MG"M1 FROM E TO A"; JP #FIND_M3
  JP#FIND_M1+7,((_IHF0+4)<>_IA)|(_IHF2<>2); HSF=A; MG"M1 FROM F TO A"; JP #FIND_M3
  JP#FIND_M1+8,((_IHG0+4)<>_IA)|(_IHG2<>2); HSG=A; MG"M1 FROM G TO A"; JP #FIND_M3
  JP#F_M1_DN,((_IHH0+4)<>_IA)|(_IHH2=0); HSH=A; MG"M1 FROM H TO A"; JP #FIND_M3
#F_M1_DN; MG"CAN'T FIND M1!"; AB
```

HX (Halt Execution)

[Program Flow]

DESCRIPTION:

The HX command halts the execution of any of the programs that may be running independently via multitasking. The parameter n specifies the program to be halted.

ARGUMENTS: HX n where

n is 0 to 7 to indicate the task number

USAGE:

While Moving	Yes	Default Value	n/a
In a Program	Yes	Default Format	n/a
Command Line	Yes		
Can be Interrogated	No		
Used as an Operand	Yes		

OPERAND USAGE:

When used as an operand, _HX n contains the running status of thread n with:

0 Thread not running

1 Thread is running

2 Thread has stopped at trippoint

RELATED COMMANDS:

XQ (Execute Program)

EXAMPLES: (assuming the file contains the label #A and # B)

XQ #A	Execute program #A, thread zero
XQ #B,2	Execute program #B, thread two
HX0	Halt thread zero
HX2	Halt thread two

IA (Internet Address)

[Setting]

DESCRIPTION:

The IA command assigns the controller an IP address.

The IA command may also be used to specify the time out value. This is only applicable when using the TCP/IP protocol.

The IA command can only be used via RS-232. Since it assigns an IP address to the controller, communication with the controller via internet cannot be accomplished until after the address has been assigned.

ARGUMENTS: IA ip0,ip1, ip2, ip3 or IA n or IA<t where

ip0, ip1, ip2, ip3 are 1 byte numbers separated by commas and represent the individual fields of the IP address.

n is the IP address for the controller which is specified as an integer representing the signed 32 bit number (two's complement).

<t specifies the time in update samples between TCP retries.

USAGE:

While Moving	No	Default Value	n = 0, t=250
In a Program	Yes	Default Format	---
Command Line	Yes		

OPERAND USAGE:

_IA0 contains the IP address representing a 32 bit signed number (Two's complement)

_IA1contains the value for t (retry time)

_IA2 contains the number of available handles

_IA3 contains the number of the handle using this operand where the number is 0 to 7. 0 represents handle A, 1 handle B, etc.

_IA4 reports the last handle that had a TCP error.

RELATED COMMANDS:

IH (Internet Handle)

EXAMPLES:

IA 151, 12, 53, 89	Assigns the controller with the address 151.12.53.89
IA 2534159705	Assigns the controller with the address 151.12.53.89
IA < 500	Sets the timeout value to 500msec

IF

[Program Flow]

DESCRIPTION:

The IF command is used in conjunction with an ENDIF command to form an IF conditional statement. The arguments are one or more conditional statements. If the conditional statement(s) evaluates true, the command interpreter will continue executing commands which follow the IF command. If the conditional statement evaluates false, the controller will ignore commands until the associated ENDIF command OR an ELSE command occurs in the program. The conditional statements MUST be enclosed on parentheses for the expression to be evaluated correctly. See the example below.

ARGUMENTS: IF condition where

Conditions are tested with the following logical operators:

<	less than	>=	greater than or equal to
>	greater than	<>	not equal
=	equal to	|	logical OR (pipe symbol)
<=	less than or equal to	&	logical AND

USAGE:

While Moving	Yes	Default Value	---
In a Program	Yes	Default Format	---
Command Line	No		

RELATED COMMANDS:

ELSE Optional command used only after IF command

ENDIF End of IF conditional Statement

EXAMPLES:

IF (_TEX<1000)	IF conditional statement based on X motor position
MG "Motor within 1000 counts of zero"	Message to be executed if “IF” conditional statement
ENDIF	End of IF conditional statement
IF ((TEMP=126) | (TEMP=123))	
JS # RELEASE	
JS # ASSIGN	
JS # CONNECT	
ENDIF	

IH (Internet Handle)

[Setting]

DESCRIPTION:

The IH command is used when the SMC–4000 must initiate communication to another Ethernet device. This command opens a handle and connects to a slave.

The controller may have 8 handles open at any given time. They are designated by the letters A through H. To open a handle, the user must specify:

The IP address of the slave

The type of session: TCP/IP or UDP/IP

The port number of the slave. This number is not necessary if the slave device doesn't require a specific port value. If not specified, the controller specifies the port value as 502.

ARGUMENTS: IHh= ip0, ip1, ip2, ip3 <p > q or IHh=n <p > q orIHh= >r where

Argument	Minimum	Maximum	Note
h	A	H	Internet handle
ip0 - ip3	0	255	Four bytes of IP address separated by commas
n	-2147483648	2147483647	32 bit address alternative to ip0 - ip3
S=>C	-1 (UDP)	-2 (TCP)	Close the handle that sent the command
T=>C	-1 (UDP)	-2 (TCP)	Close handles except the one sending the command
<p	0	65535	Specifies the port number of the slave, not required for opening a handle
>q	0	2	Set connection type; 0=none; 1=UDP; 2=TCP
>r	-1 (UDP)	-2 (TCP)	Terminate connection, and handle to be freed

OPERAND USAGE:

_IHh0 contains the IP address as a 32 bit number

_IHh1 contains the slave port number

_IHh2 contains a 0 if the handle is free

contains a 1 if it is for a UDP slave

contains a 2 if it is for a TCP slave

contains a -1 if it is for a UDP master

contains a -2 if it is for a TCP master

contains a -5 if attempting to connect by UDP

contains a -6 if attempting to connect by TCP

_IHh3 contains a 0 if the ARP was successful

contains a 1 if it has failed or is still in progress.

_IHh4 contains a 1 if the SA command is waiting for acknowledgement from a slave

contains a 2 if the SA command received a colon

contains a 3 if the SA command received a question mark

contains a 4 if the SA command timed out

USAGE:

While Moving	Yes	Default Value	----
In a Program	Yes	Default Format	
Command Line	Yes		

RELATED COMMANDS:

IA (Internet Address)

EXAMPLES:

IHA=251,29,51,1	Open handle A at IP address 251.29.51.1
IHA= -2095238399	Open handle A at IP address 251.29.51.1

***Note When the IH command is given, the controller initializes an ARP on the slave device before opening a handle. This operation can cause a small time delay before the controller responds.**

II (Input Interrupt)

[Configuration]

DESCRIPTION:

The II command enables the interrupt function for the specified inputs. This function triggers when the controller sees a logic change from high to low on a specified input.

If the #ININT special label is included in the program and any of the specified inputs go low during program execution, the program will jump to the subroutine with label #ININT. Any trippoints set by the program will be cleared but can be re-enabled by the proper termination of the interrupt subroutine using RI.

To avoid returning to the main program on an interrupt, use the ZS (Zero Subroutine Stack) command to zero the subroutine stack and use the II command to re-enable the interrupt.

ARGUMENTS: II m,n,o,p are integers where

Argument	Min	Max	Note	Example	Meaning
m	0	8	Zero disables the interrupts, otherwise, specify the input number. If parameter n will be used, the value of "m" specifies the lowest input number to be used for the input interrupt.	II 3	Input #3 will cause an interrupt when it goes low.
n	2	8	Optional argument used with "m" to specify a range of inputs. When the "n" argument is omitted, only the input specified by the "m" parameter will be enabled.	II 2, 4	Input #2, Input #3, and Input #4 are enabled for interrupt.
o	1	255	This argument is an alternative to specifying a range of inputs. Specify the inputs that are enabled for interrupt in a binary format. (If "m" and "n" are specified, "o" will be ignored.)	II,, $0F	Equivalent to binary 00001111, inputs #1 through #4 will be enabled for interrupt.
p	1	255	Specifies interrupts that should activate with logic one. Specify the inputs that are logic one in a binary format. This argument logically ANDs with inputs already specified in the above arguments.	II 1, 4,, 2	"p"equivalent is 00000010, so only Input #2 (of #1 through #4) will interrupt active high. 1,3, and 4 will interrupt active low.

USAGE:

While Moving	Yes	Default Value	n/a
In a Program	Yes	Default Format	3.0 (mask only)
Command Line	No		
Can be Interrogated	Yes		
Used as an Operand	Yes		

RELATED COMMANDS:

RI (Return from Interrupt)

#ININT Interrupt Subroutine Special label

AI (After Input)

EXAMPLES:

#A	Program A
II ,,,64	Specify interrupt on input #7 going high not implemented.
JG 5000	Specify jog speed
BG	Begin motion
#LOOP; JP #LOOP	Loop
EN	End Program
#ININT	Interrupt subroutine
ST; MG "INTERRUPT"	Stop X, print message
AM	After stopped
#CLEAR; JP#CLEAR,@IN[1]=0	Check for interrupt clear
BG	Begin motion
RI	Return to main program

IL (Integrator Limit)

[Tuning]

DESCRIPTION:

The IL command limits the effect of the integrator function in the filter to a certain voltage. For example, IL 2 limits the output of the integrator to the +/-2 Volt range. This is very effective in allowing higher KI (Integrator) values without adding instability.

A negative parameter also freezes the effect of the integrator during a move. For example, IL -3 limits the integrator output to +/-3V. If, at the start of motion, the integrator output is 1.6 Volts, that level will be maintained through the move. Note, however, that the KD (Derivative Constant) and KP (Proportional Constant) terms remain active in any case.

ARGUMENTS: IL x, y, z, w or ILX=x or IL a, b, c, d where

x, y z, w, or a, b, c, d are unsigned integers

USAGE:

While Moving	Yes	Minimum Value	-10
In a Program	Yes	Maximum Value	10
Command Line	Yes	Default Value	10 (disabled)
Can be Interrogated	Yes	Default Format	1.4
Used as an Operand	Yes		
Virtual Axis	No		

OPERAND USAGE:

_ILn contains the value of the integrator limit in volts where n is an axis letter.

RELATED COMMANDS:

KI (Integrator)

EXAMPLES:

KI 2	Integrator constants
IL 3	Integrator limits
IL ?	Returns the limit
3.0000	

IN (Input Variable)

[General]

DESCRIPTION:

The IN command allows a variable to be input from the serial port. An optional prompt message can be displayed. The variable value must be followed by a carriage return. The entered value is assigned to the specified variable name.

The IN command holds up execution of following commands in the program thread until a carriage return or semicolon is entered. If no value is given prior to a semicolon or carriage return, the previous variable value is kept. Input Interrupts, Error Interrupts and Limit Switch Interrupts will still be active.

ARGUMENTS: IN{P1} "m" , n {So} where

"m" is the prompt message. May be letters, numbers, or symbols up to maximum line length and must be placed in quotations.

n is the name of variable to store the new value in.

{P1} specifies the port, if omitted, the default port is assumed.

{So} specifies string data where o is the number of characters from 1 to 6

***Note 1: The IN command defaults to {P1}, and must only be used with the serial port.**
***Note 2: IN command can only be used in thread 0.**

USAGE:

While Moving	Yes	Default Value	
In a Program	Yes	Default Format	Position Format
Command Line	No		
Can be Interrogated	No		
Used as an Operand	No		

EXAMPLES: Operator specifies material length in inches and speed in inches/sec (2 pitch lead screw, 2000 counts/rev encoder).

#A	Program A
CI -1	Clear Input Buffer
IN "Enter Speed (in/sec)",V1	Prompt operator for speed
IN "Enter Length(in)",V2	Prompt for length
V3=V1*4000	Convert units to counts/sec
V4=V2*4000	Convert units to counts
SP V3	Speed command
PR V4	Position command
BGX;AMX	Begin motion; Wait for motion complete
MG "MOVE DONE"	Print Message
EN	End Program

@IN (Input)

[I/O]

DESCRIPTION:

@IN returns the status of the digital input number or variable given in square brackets. Note that the @IN command is a function, which means that it does not follow the convention of the commands, and does not require the underscore when used as an operand.

ARGUMENTS: @IN [n] where

n is an integer corresponding to a specific input to be read.

MODBUS:

***Note With Modbus devices, I/O points of the devices are calculated using the following formula:**

n = (SlaveAddress*1000) + (HandleNum*1000) + ((Module-1)*4) + (Bitnum-1)

Slave Address is used when ModBus device has slave devices connected to it and specified as Addresses 0 to 255. Note that the use of slave devices for modbus are very rare and this number will usually be 0.

HandleNum is the handle specifier from A to H (1 - 8).

Module is the position of the module in the rack from 1 to 16.

BitNum is the I/O point in the module from 1 to 4.

USAGE:

While Moving	Yes	Minimum n value	1
In a Program	Yes	Maximum n value	8
Not in a program	Yes	Default Value	---
Can be Interrogated	No	Default Format	---
Used as an Operand	Yes		

EXAMPLES:

#TEST	Program TEST
VAR1=2	Set variable
MG @IN[VAR1]	Display the status of digital input 2
VAR2=@IN[VAR1]+1	Perform calculation
EN	End of program
IF(@IN[604])	
MG”Slave input 4 is ON”	
ELSE	
MG”Slave input 4 is OFF”	
ENDIF	

@INT (Integer)

[Function]

DESCRIPTION:

@INT returns only the whole number part of a number or variable given in square brackets. Note that the @INT command is a function, which means that it does not follow the convention of the commands, and does not require the underscore when used as an operand.

ARGUMENTS: @INT [n] ***where***

n is a number

USAGE:

While Moving	Yes	Minimum n value	-2147483648.9999
In a Program	Yes	Maximum n value	2147483648.9999
Not in a program	Yes	Default Value	---
Can be Interrogated	No	Default Format	---
Used as an Operand	Yes		

EXAMPLES:

#TEST	Program TEST
VAR1=123.456	Set variable
MG @INT[VAR1]	Display only the whole number portion of VAR1
VAR2=@INT[VAR1]+25	Perform calculation
EN	End of program

IP (Increment Position)

[Motion]

DESCRIPTION:

The IP command allows for an update in the commanded position while the motor is moving. This command does not require a BG (Begin) command. The command has three effects depending on the motion being executed. The units of this command are quadrature counts.

Case 1: Motor is standing still

An IP n command is equivalent to a PR (Position Relative) n and BG (Begin) command. The motor will move to the specified position at the requested slew speed and acceleration.

Case 2: Motor is moving towards specified position

An IP n command will cause the motor to move to a new position target, which is the old target plus n. n must be in the same direction as the existing motion (final target cannot be closer).

Case 3: Motor is in Jog Mode

An IP (Increment Position) n command will cause the motor to instantly try to servo to a position n from the present instantaneous position. The SP (Speed) and AC (Acceleration) parameters have no effect. This command is useful for making small corrections when synchronizing 2 axes in which one of the axis' speed is indeterminate due to a variable diameter pulley.

Warning: When the motor is in jog mode, an IP (Increment Position) command will create an instantaneous position error. In this mode, the IP should only be used to make small incremental position movements.

ARGUMENTS: IP x, y, z, w or IPX=x or IP a, b, c, d where

x, y z, w, or a, b, c, d are unsigned integers

USAGE:

While Moving	Yes	Minimum Value	-2147483648
In a Program	Yes	Maximum Value	2147483647
Command Line	Yes	Default Value	n/a
Can be Interrogated	Yes	Default Format	10.0
Used as an Operand	No		
Virtual Axis	No		

EXAMPLES:

:IP 50	50 counts with set acceleration and speed
#CORRECT	Label
AC 100000	Set acceleration
JG 10000;BG	Jog at 10000 counts/sec rate
WT 1000	Wait 1000 msec
IP 10	Move the motor 10 counts instantaneously
ST	Stop Motion

IT (Independent Time Constant)

[Motion]

DESCRIPTION:

The IT command filters the acceleration and deceleration functions in independent moves of JG (Jog), PR (Position Relative), and PA (Position Absolute) type to produce a smooth velocity profile. The resulting profile, known as an S-curve, has continuous acceleration and results in reduced mechanical vibrations. IT sets the bandwidth of the filter where 1 means no filtering and 0.004 means maximum filtering. Note that the filtering results in longer motion time.

The use of IT will not effect the trippoints AR (After Relative) and AD (After Distance). The trippoints AR and AD monitor the profile prior to the IT filter and therefore can be satisfied before the actual distance has been reached if IT is NOT 1.

An IT value less then 1 will make the move take longer. This can be compensated for by increasing the acceleration and deceleration paraemters

ARGUMENTS: IT x, y, z, w or ITX=x or IT a, b, c, d where

n is a positive number with a resolution of 1/256

USAGE:

While Moving	Yes	Minimum Value	0.004
In a Program	Yes	Maximum Value	1.000
Command Line	Yes	Default Value	1.0
Can be Interrogated	Yes	Default Format	1.4
Used as an Operand	Yes		
Virtual Axis	Yes		

OPERAND USAGE:

_ITn will return the value of the independent time constant where n is an axis letter.

EXAMPLES:

IT 0.8	Set independent time constants
MG_ITX	Return independent time constant
0.8	

JG (Jog)

[Motion]

DESCRIPTION:

The JG command sets a speed in jog mode. The parameters following the JG set the slew speed and direction of motion. Use of the question mark returns the previously entered value or default value. The units of this are counts/second. The AC (Acceleration) and DC (Deceleration) commands work in this mode.

ARGUMENTS: JG x, y, z, w or JGX=x or JG a, b, c, d where

x, y z, w, or a, b, c, d are signed integers

USAGE:

While Moving	Yes	Minimum Value	-12,000,000
In a Program	Yes	Maximum Value	12,000,000
Command Line	Yes	Default Value	25000
Can be Interrogated	Yes	Default Format	Position Format
Used as an Operand	Yes		
Virtual Axis	Yes		

OPERAND USAGE:

_JGn will return the absolute value of the jog speed in counts per second where n is an axis letter.

RELATED COMMANDS:

BG (Begin)

ST (Stop)

AC (Acceleration)

DC (Deceleration)

IP (Increment Position)

TV (Tell Velocity)

EXAMPLES:

JG 100	Set for jog mode with a slew speed of 100 counts/sec
BG	Begin Motion
JG -2000	Change speed and direction.

JP (Jump to Program Location)

[Program Flow]

DESCRIPTION:

The JP command causes a jump to a program location on a specified condition (optional). The program location may be any label. The condition is a conditional statement which uses a logical operator such as equal to or less than. A jump is executed if the specified condition is true.

Multiple conditions can be used in a single jump statement. Conditional statements are combined in pairs using operands “&” and “|”. The “&” operand between two conditions requires both statements to be true for the combined statement to be true. The ”|” operand between two conditions requires that one statement be true for the combined statement to be true.

***Note Each condition must be in parenthesis for controller evaluation as a boolean expression.**

ARGUMENTS: JP location, condition where

location is a program label

condition is a conditional statement using a logical operator

The logical operators are:

<	less than	>=	greater than or equal to
>	greater than	<>	not equal
=	equal to	|	logical OR (pipe symbol)
<=	less than or equal to	&	logical AND

USAGE:

While Moving	Yes	Default Value	n/a
In a Program	Yes	Default Format	n/a
Command Line	No		
Can be Interrogated	No		
Used as an Operand	No		

EXAMPLES:

JP #POS1,V1<5	Jump to label #POS1 if variable V1 is less than 5
JP #A,V7*V8=0	Jump to #A if V7 times V8 equals 0
JP #B	Jump to #B (no condition)

HINT: JP is similar to an IF, THEN command. Text to the right of the comma is the condition that must be met for a jump to occur. The destination is the specified label before the comma.

JS (Jump to Subroutine)

[Program Flow]

DESCRIPTION:

The JS command will change the sequential order of execution of commands in a program. If the jump is executed, the program will continue at the label specified by the destination parameter. The line number of the JS command is saved and after the next EN (End) command is encountered, program execution will continue with the instruction following the JS command. The JS command can be nested 16 deep.

Multiple conditions can be used in a single jump subroutine statement. The conditional statements are combined in pairs using the operands "&" and "|". The "&" operand between any two conditions requires that both statements must be true for the combined statement to be true. The "|" operand between any two conditions requires that only one statement be true for the combined statement to be true.

***Note Each condition must be placed in parenthesis for proper evaluation by the controller as a boolean expression. Subroutines can be nested 16 deep in the standard controller.**

ARGUMENTS: JS destination,condition where

destination is a line number or label

condition is a conditional statement using a logical operator

The logical operators are:

<	less than	>=	greater than or equal to
>	greater than	<>	not equal
=	equal to	|	logical OR (pipe symbol)
<=	less than or equal to	&	logical AND

USAGE:

While Moving	Yes	Default Value	---
In a Program	Yes	Default Format	---
Command Line	No		
Can be Interrogated	No		
Used as an Operand	No		

RELATED COMMANDS:

EN (End)

EXAMPLES:

JS #SQUARE,V1<5	Jump to subroutine #SQUARE if V1 is less than 5
JS #LOOP,V1<>0	Jump to #LOOP if V1 is not equal to 0
JS #A	Jump to subroutine #A (no condition)

KD (Derivative Constant)

[Tuning]

DESCRIPTION:

KD designates the derivative constant in the controller filter. The filter transfer function is

D(z) = 4 * KP + 4 * KD(z-1)/z + KIz/2 (z-1)

For further details on the filter see the section Theory of Operation.

ARGUMENTS: KD x, y, z, w or KDX=x or KD a, b, c, d where

x, y z, w, or a, b, c, d are unsigned integers

USAGE:

While Moving	Yes	Minimum Value	0
In a Program	Yes	Maximum Value	4095.875
Command Line	Yes	Default Value	10
Can be Interrogated	Yes	Default Format	4.2
Used as an Operand	Yes		
Virtual Axis	No		

OPERAND USAGE:

_KDn contains the value of the derivative constant where n is an axis letter.

RELATED COMMANDS:

JP (Jump to Program Location)

KI (Integrator)

EXAMPLES

KD 100	Specify KD
MG_KDX	Return KD
0100.00	

KI (Integrator)

[Tuning]

DESCRIPTION:

The KI command sets the integral gain of the control loop. It fits in the control equation as follows:

D(z) = 4 * KP + 4 * KD(z-1)/z + KI z/2(z-1)

The integrator term will reduce the position error at rest to zero.

ARGUMENTS: KI x, y, z, w or KIX=x or KI a, b, c, d where

x, y z, w, or a, b, c, d are unsigned integers

USAGE:

While Moving	Yes	Minimum Value	0
In a Program	Yes	Maximum Value	2047.875
Command Line	Yes	Default Value	0
Can be Interrogated	Yes	Default Format	4.0
Used as an Operand	Yes		
Virtual Axis	No		

OPERAND USAGE:

_KIn contains the value of the integrator where n is an axis letter.

RELATED COMMANDS:

KP (Proportional Constant)

KD (Derivative Constant)

IL (Integrator Limit)

EXAMPLES:

KI 12	Specify integral
MG_KIX	Return value
0012	

KP (Proportional Constant)

[Tuning]

DESCRIPTION:

KP designates the proportional constant in the controller filter. The filter transfer function is

D(z) = 4 * KP + 4 * KD(z-1)/z + KI z/2(z-1)

For further details see the section Theory of Operation.

ARGUMENTS: KP x, y, z, w or KPX=x or KP a, b, c, d where

x, y z, w, or a, b, c, d are unsigned integers

USAGE:

While Moving	Yes	Minimum Value	0
In a Program	Yes	Maximum Value	1023.875
Command Line	Yes	Default Value	1
Can be Interrogated	Yes	Default Format	4.2
Used as an Operand	Yes		
Virtual Axis	No		

OPERAND USAGE:

_KPn contains the value of the proportional constant where n is an axis letter.

RELATED COMMANDS:

KP (Proportional Constant)

KI (Integrator)

KS (Step Motor Smoothing)

[Motion]

DESCRIPTION:

The KS parameter sets the amount of smoothing of stepper motor pulses. This is most useful when operating in full or half step mode. Larger values of KS provide greater smoothness. This parameter will also increase the motion time by 3KS sampling periods. KS adds a single pole low pass filter onto the output of the motion profiler.

Note: Note: KS will cause a delay in the generation of output steps.

ARGUMENTS: KS n,n,n,n or KSA=n where

n is a positive number in the range between .5 and 16 with a resolution of 1/32.

USAGE:

While Moving	Yes	Default Value	1.313
In a Program	Yes	Default Format	4.0
Command Line	Yes		
Virtual Axis	No		

OPERAND USAGE:

_KSn contains the value of the derivative constant for the specified axis.

RELATED COMMANDS:

MT (Motor Type)

EXAMPLES:

KS 2,4,8	Specify a,b,c axes
KS 5	Specify a-axis only
KS ,,15	Specify c-axis only

NOTE: KS is valid for step motor only.

LA (List Arrays)

[General]

DESCRIPTION:

The LA command returns a list of all arrays in memory. The listing will be in alphabetical order. The size of each array will be included next to each array name in square brackets.

ARGUMENTS: None

USAGE:

While Moving	Yes	Default Value	-
In a Program	Yes	Default Format	-
Command Line	Yes		

RELATED COMMANDS:

LL (List Labels)

LS (List Program)

LV (List Variables)

EXAMPLES:

: LA	
CA [10]	
LA [5]	
NY [25]	
VA [17]	

LC (Lock Controller)

[Configuration]

DESCRIPTION:

The (LC) Lock Controller command is used to prohibit the execution of certain commands from the serial port by setting a security password. See the table below for a list of commands that are disabled in the “Locked”mode. When this command is successfully executed, it automatically burns the new configuration into the EEPROM. All parameters listedon the BN (Burn Parameters) page are stored when LC is succesfully executed.

ARGUMENTS: LC p, l where

p is the password as previously established with the "PW" command.

"l" is the Lock setting, 0=Unlock,

1	Lock commands (see table),
2	Lock commands and prohibit setting any commands from the serial port.
3	Allow download but lock PW, UL, TR, LS, and ED

USAGE:

While Moving	Yes	Default Value	---
In a Program	No	Default Format	---
Command Line	Yes		
Used as an Operand	Yes		
Can be Interrogated	Yes		

OPERAND USAGE:

_LC will return the lock state of the controller, 0 = not locked, 1 = specific commands locked, 2 = All commands locked including from serial port and ethernet port except the LC command.

RELATED COMMANDS:

PW (Password)

COMMANDS DISABLED WHILE LOCKED = 1:

BN (Burn Parameters)	TR (Trace Mode)
BP (Burn Program)	DL (Download)
BV (Burn Variables)	LS (List Program)
UL (Upload)	ED (Edit Mode)
IA (Internet Address)	PW (Password)

EXAMPLES:

LC apple,2	Locks controller, assuming the valid password is “apple.”
BN	Burn command (invalid when locked)
?	Receive question mark
TC1	Tell Code returns “Command not valid while controller is locked.”

LE (Linear Interpolation End)

[Motion]

DESCRIPTION: LE

Signifies the end of a linear interpolation sequence. It follow the last LI (Linear Interpolation Distance) specification in a linear sequence. After the LE specification, the controller issues commands to decelerate the motor to a stop. The VE (Vector Sequence End) command is interchangeable with the LE command.

The LE command will apply to the selected coordinate system, S or T. To select the coordinate system, use the command CAS or CAT.

ARGUMENTS:

None

USAGE:

While Moving	Yes	Default Value	---
In a Program	Yes	Default Format	---
Command Line	Yes		

OPERAND USAGE:

_LEn contains the total vector move length in encoder counts.

RELATED COMMANDS:

CA (Coordinate Axes)

LI (Linear Interpolation Distance)

BG (Begin)

LM (Linear Interpolation Mode)

VS (Vector Speed)

VA (Vector Acceleration)

VD (Vector Deceleration)

PF (Position Format)

EXAMPLES:

CAS	Specify S coordinated motion system
LM CD	Specify linear interpolation mode for C and D axes
LI ,,100,200	Specify linear distance
LE	End linear move
BGS	Begin sequence

_LF* (Forward Limit)

[Status]

DESCRIPTION: _LF XYZW or LF ABCD

The _LF operand contains the state of the forward limit switch. A value of zero always indicates that the limit is active, no matter what configuration the CN command is set to.

***Note** **Note: This is not a command.**

EXAMPLES:

MG _LFX	Display the status of the forward limit switch
JP#A,_LFX=0	Jump to label, #A, forward limit switch is activated

LI (Linear Interpolation Distance)

[Motion]

DESCRIPTION:

The LI a,b,c,d command specifies the incremental distance of travel for each axis in the Linear Interpolation (LM) mode. LI parameter are relative distances given with respect to the current axis positions. Up to 511 LI specifications may be given ahead of the Begin Sequence (BGS) command. Additional LI commands may be sent during motion when the controller sequence buffer frees additional spaces for new vector segments. The LE (Linear Interpolation End) command must be given after the last LI specification in a sequence, it causes deceleration to a stop at the last LI command. It is the responsibility of the user to keep enough LI segments in the controller sequence buffer to ensure continuous motion.

LM (Linear Interpolation Mode) returns the available spaces for LI segments that can be sent to the buffer. 511 returned means the buffer is empty and 511 LI segments can be sent. A zero means the buffer is full and no additional segments can be sent. It should be noted that the controller computes the vector speed based on the axes specified in the LM mode. For example, LM ABC designates linear interpolation for the A, B, and C axes.The speed of these axes will be computed from $VS^2=AS^2+BS^2+CS^2$, where AS, BS and CS are the speed of the A, B, and C axes. If the LI command specifies only A and B, the speed of C will still be used in the vector calculations. The controller always uses the axis specifications from LM, not LI, to compute the speed. The parameter n is optional and can be used to define the vector speed that is attached to the motion segment.

The LI command will apply to the selected coordinate system, S or T. To select the coordinate system, use the command CAS or CAT.

ARGUMENTS: LI n,n,n,n, <o> p or LIA=n where

n is a signed integers in the range -8,388,607 to 8,388,607 and represent incremental move distance.

o specifies a vector speed to be taken into effect at the execution of the linear segment. s is an unsigned even integer between 0 and 12,000,000 for servo motor operation and between 0 and 3,000,000 for stepper motors.

p specifies a vector speed to be achieved at the end of the linear segment. Based on vector accel and decel rates, o is an unsigned even integer between 0 and 8,000,000.

Argument	Min	Max	Note	Example	Meaning
n	-8388607	8388607	The incremental move distance.	LI 500, 200	500 encoder count move on the X axis, 200 count move on the Y axis.
o	0	12000000	Vector speed to be taken into effect at the execution of this segment.	LI 500 <40000	500 encoder count move on the X axis. Change to a vector speed of 40000 counts per second.
p	0	8000000	Vector speed to be taken into effect at the end of this segment.	LI 500 >40000	500 encoder count move on the X axis. Change to a vector speed of 40000 counts per second at the end of the segment.

USAGE:

While Moving	Yes	Default Value	-
In a Program	Yes	Default Format	-
Command Line	Yes		
Controller Usage	All controllers		

RELATED COMMANDS:

LE (Linear Interpolation End)

BG (Begin)

LM (Linear Interpolation Mode)

CS (Clear Sequence)

VS (Vector Speed)

VA (Vector Acceleration)

VD (Vector Deceleration)

EXAMPLES:

LM ABC	Specify Linear Interpolation Mode
LI 1000,2000,3000	Specify distance
LE	Last segment
BGS	Begin sequence

LL (List Labels)

[General]

DESCRIPTION:

The LL command returns a listing of all of the program labels in memory. The listing will be in alphabetical order and will include the line numbers.

ARGUMENTS: None

USAGE:

While Moving	Yes	Default Value	---
In a Program	Yes	Default Format	---
Command Line	Yes		
Can be Interrogated	Yes		
Used as an Operand	Yes		

RELATED COMMANDS:

LV (List Variables)

LA (List Arrays)

EXAMPLES:

: LL	
# FIVE = 21	
# FOUR = 3	
# ONE = 16	
# THREE = 42	
# TWO = 0	

LM (Linear Interpolation Mode)

[Setting]

DESCRIPTION:

The LM command specifies the linear interpolation mode and specifies the axes for linear interpolation. Any set of 1 thru 4 axes may be used for linear interpolation. LI commands are used to specify the travel distances for linear interpolation. TheLE (Linear Interpolation End) command specifies the end of the linear interpolation sequence. Several LI (Linear Interpolation Distance) commands may be given as long as the controller sequence buffer has room for additional segments. Once the LM command has been given, it does not need to be given again unless the VM (Coordinated Motion Mode) command has been used.

It should be noted that the controller computes the vector speed based on the axes specified in the LM mode. For example, LM ABC designates linear interpolation for the A,B, and C axes. The speed of these axes will be computed from $VS^2=AS^2+BS^2+CS^2$, where AS, BS and CS are the speed of the A, B, and C axes. In this example, if the LI command specifies only A and B, the speed of C will still be used in the vector calculations. The controller always uses the axis specifications from LM, not LI, to compute the speed.

The LM command will apply to the selected coordinate system, S or T. To select the coordinate system, use the command CAS or CAT.

ARGUMENTS: LM nnnn where

n is A,B,C,D or any combination to specify the axis or axes

USAGE:

While Moving	Yes	Default Value	---
In a Program	Yes	Default Format	---
Command Line	Yes		
Virtual Axis	No		

OPERAND USAGE:

_LMn contains the number of spaces available in the sequence buffer for the “n” coordinate system, S or T.

RELATED COMMANDS:

LE (Linear Interpolation End)

LI (Linear Interpolation Distance)

VA (Vector Acceleration)

VS (Vector Speed)

VD (Vector Deceleration)

AV (After Vector Distance)

CS (Clear Sequence)

EXAMPLES:

LM ABCD	Specify linear interpolation mode
VS 10000; VA 100000;VD 1000000	Specify vector speed, acceleration and deceleration
LI 100,200,300,400	Specify linear distance
LI 200,300,400,500	Specify linear distance
LE; BGS	Last vector, then begin motion

_LR* (Reverse Limit)

[Status]

DESCRIPTION: _LR XYZW or ABCD

*The _LR operand contains the state of the reverse limit switch. A value of zero always indicates that the limit is active no matter what the configuration of the CN command is.

***Note** **Note: This is not a command.**

EXAMPLES:

MG _LRX	Display the status of the reverse limit switch
JP#A,_LRX=0	Jump to label, #A, when reverse limit switch is activated

LS (List Program)

[General]

DESCRIPTION:

The LS command sends a listing of the program memory out of the port that issued the command. The listing will start with the line pointed to by the first parameter, which can be either a line number or a label. If no parameter is specified, it will start with line 0. The listing will end with the line pointed to by the second parameter--again either a line number or label. If no parameter is specified, the listing will go to the last line of the program.

ARGUMENTS: LS n,m where

n,m are valid numbers from 0 to 2000, or labels. n is the first line to be listed, m is the last.

_LS returns the line number the program will return to after the current subroutine ends. If a program is not running, the value is negative and reports the number of program lines in the controller.

USAGE:

While Moving	Yes	Default Value	0,Last Line
In a Program	No	Default Format	---
Command Line	Yes		
Can be Interrogated	No		
Used as an Operand	No		

EXAMPLES:

:LS #A,6	List program starting at #A through line 6
002 #A	
003 PR 500	
004 BG	
005 AM	
006 WT 200	

HINT: Remember to quit the Edit Mode <cntrl> Q prior to giving the LS command.

LT (Latch Target)

[Motion]

DESCRIPTION:

The LT command is used for stopping an axis a defined distance after a registration mark (latch) input. The distance specified by the LT command is in encoder counts. The distance must be sufficiently large for the controller to decelerate normally at the specified deceleration rate. A stop code will be generated if the distance is too small to stop for the deceleration rate or if the speed is too high. To Disable the latch target, set LTX=0.

ARGUMENTS: LTX=x LTx,y,z,w LTa,b,c,d Where abcdxyzw are integers

POSSIBLE STOP CODES:

1 Motors stopped at commanded independent position (Latch input not received)

40 Stopped at Latch Target.

41 Latch Target overrun due to limit switch or stop command.

42 Latch Target overrun due to insufficient distance.

USAGE:

While Moving	Yes	Default Value	n/a
In a Program	Yes	Default Format	---
Command Line	Yes		
Can be Interrogated	Yes		
Used as an Operand	Yes		
Virtual Axis	No		

RELATED COMMANDS:

AL (Arm Latch)

RL (Report Latch)

EXAMPLES:

ALX	Set latch function
LTX=25000	Set Latch Target to stop 25000 counts after registration
PRX=100000	Position Relative Move
BGX	Begin Motion
AMX	After Motion
JP #NOMRK,_SCX=1	Jump to #NOMARK routine if did not receive a registration mark

LV (List Variables)

[General]

DESCRIPTION:

The LV command returns a listing of all of the user variables in memory. The listing will be in alphabetical order.

ARGUMENTS: None

USAGE:

While Moving	Yes	Default Value	n/a
In a Program	Yes	Default Format	n/a
Command Line	Yes		
Can be Interrogated	Yes		
Used as an Operand	No		

RELATED COMMANDS:

LL (List Labels)

EXAMPLES:

: LV	
APPLE = 60.0000	
BOY = 25.0000	
ZEBRA = 37.0000	

LZ (Leading Zeros)

[Setting]

DESCRIPTION:

The LZ command is used for formatting the values returned from interrogation commands or interrogation of variables and arrays. By enabling the LZ function, all leading zeros of returned values will be removed. This will reduce transmission time and potentially ease formatting issues on connected devices.

ARGUMENTS: LZ n where

1 to remove leading zeros

0 to disable the leading zero removal

USAGE:

While Moving	Yes	Default Value	0
In a Program	Yes	Default Format	n/a
Command Line	Yes		
Can be Interrogated	Yes		
Used as an Operand	Yes		

OPERAND USAGE

_LZ returns the state of the LZ (Leading Zeros) command.

EXAMPLES:

TE	Tell error
0004	
LZ 1	Inhibit leading zeros
TE	Tell error
4	

MB (Modbus)

[I/O]

DESCRIPTION:

The MB command is used to communicate with I/O devices using the first two levels of the Modbus protocol.

The format of the command varies depending on each function code. The function code, -1, designates that the first level of Modbus is used (creates raw packets and receives raw data). The other codes are the 10 major function codes of the second level that the SMC–4000 supports.

***Note For those command formats that have "addr", this is the slave address. The slave address may be designated or defaulted to the device handle number.**

***Note All formats contain an h parameter. This designates connection handle number (A thru H).**

ARGUMENTS:

Function	Meaning	Example
-1	Raw Packets	MBh = -1, y, array [ ]
1	Read Coil Status	MBh = a, 1, t, b, array [ ]
2	Read Input Status	MBh = a, 2 , t, b, array [ ]
3	Read Holding Registers	MBh = a, 3, e, r, array [ ]
4	Read Input Registers	MBh = a, 4, e, r, array [ ]
5	Write Single Coil	MBh = a, 5, t, c
6	Write Single Register	MBh = a, 6,g, s
7	Read Exception Status	MBh = a, 7, array [ ]
15	Write Multiple Coils	MBh = a, 15, t, b, array [ ]
16	Write Multiple Registers	MBh = a, 16, e, r, array [ ]
17	Report Slave ID	MBh = a, 17, array [ ]

Argument	Description	Argument	Description
a	Slave address	h	Connection handle number
array [ ]	Name of array containing data	r	Number of registers
b	Number of bits	s	16 bit value
c	0 or 1 (to turn coil OFF or ON)	t	Starting bit number
e	Starting register	y	Number of bytes
g	Register number		

USAGE:

While Moving	Yes	Default Value	---
In a Program	Yes	Default Format	---
Command Line	Yes		

EXAMPLES:

IHF=>-2	Disconnect handle
IHF=192,168,3,11<502>2	Connect handle
MBF=6,16,632+(MODULE*8),NUMOFIO*2,A[ ]	Send Modbus configuration command
MBF=6,6,1025,1	Modbus command to burn parameters in OPTO-22 Ethernet module
MBF=6,2,0,1,A[ ]	Read single digital input into array A

This program was designed to read four analog inputs from 2 analog input cards (the first two cards) in the rack of a Wago I/O system. Note Modbus function 3 is used to read the four regsiters starting at register 0 (The E & R variables.) Register 0 correlates to Modbus address 40000. The data received is a binary value that reporesents a +/- 10 volt input, thus the conversion calculation.

```
#TEST
    DM WAGO[100]
    WT 1000
    IA 192,168,3,100; WT500; MW1
    IHC=>-2
    #WT; WT 50; JP#WT,_IHC2<>0
    IHC=192,168,3,211<502>2
    #WTHC; WT 50; JP #WTHC,_IHC2<>-2
    E=0; R=4; i=0

#MAIN
    MBC=3,3,E,R,WAGO[]
    #MGLP
        IF (WAGO[i]<$8000)
           VOLTS=(WAGO[i]/$7FFF)*10
        ELSE
            VOLTS=(WAGO[i]-$10000)/$7FFF*10
        ENDIF
        MG VOLTS{$F2.4}," " {N}; i=i+1
    JP #MGLP,i<R; i=0
    MG;    WT500
JP #MAIN
```

MC (Motion Complete)

[Trippoint]

DESCRIPTION:

The MC command is a trippoint used to control the timing of events. This command will hold up execution of the following commands until the current move on the specified axis or axes is completed and the encoder reaches or passes the specified position. Any combination of axes may be specified with the MC command. For example, XY waits for motion on both the A and B axis to be complete. MC with no parameter specifies that motion on all axes is complete. The TW (Time Wait) command sets the timeout to declare an error if the encoder is not in position within the specified time. If a timeout occurs, the trippoint will clear and the SC (Stop Code) command will be set to 99. An application program will jump to the special label #MCTIME, if included in your program.

When used in stepper mode, the controller will hold up execution of the proceeding commands until the controller has generated the same number of steps as specified in the commanded position. The actual number of steps that have been generated can be monitored by using the TD (Tell Dual (Auxiliary) Encoder) interrogation command. Note: The MC (Motion Complete) command is recommended when operating with stepper motors since the generation of step pulses can be delayed due to the KS (Step Motor Smoothing) function. In this case, the MC command would only be satisfied after all steps are generated.

ARGUMENTS: MC nnnn where

n is A,B,C,D,X,Y,Z,W or any combination specifing the axis or axes.

No argument specifies that motion on all axes is complete.

USAGE:

While Moving	Yes	Default Value	---
In a Program	Yes	Default Format	---
Command Line	Yes		
Virtual Axis	No		

RELATED COMMANDS:

BG (Begin)

AM (After Motion)

TW (Time Wait)

SC (Stop Code)

EXAMPLES:

#MOVE	Program MOVE
PR2000,4000	Independent Move on A and B axis
BG AB	Start the B-axis
MC AB	After the move is complete on T coordinate system
MG “DONE”; TP	Print message
EN	End of Program

Hint: ***MC can be used to verify that the actual motion has been completed. In certain applications, that have very little KI (integration), it is possible that the axis does not get to the exact position specified. This means the MC command will wait the entire time of the TW command. Set the TW command to a realistic value.***

MF (Motion Forward)

[Trippoint]

DESCRIPTION:

The MF command is a trippoint used to control the timing of events. This command will hold up the execution of the following command until the specified motor moves forward and crosses the position specified. The units of the command are in quadrature counts. The MF command can also be used when the encoder is the master and not under servo control, because the actual position is monitored.

ARGUMENTS: MFx, y, z, w or MFX=x or MFa ,b, c, d where

x, y z, w, or a, b, c, d are signed integers

USAGE:

While Moving	Yes	Minimum Value	-2147483648
In a Program	Yes	Maximum Value	2147483647
Command Line	Yes	Default Value	---
Can be Interrogated	No	Default Format	---
Used as an Operand	No		
Virtual Axis	No		

RELATED COMMANDS:

AR (After Relative)

MR (Motion Reverse)

AP (After Absolute Position)

EXAMPLES:

#TEST	Program B
DP0	Define zero
JG 1000	Jog mode (speed of 1000 counts/sec)
BG	Begin move
MF 2000	After passing the position 2000
V1=_TP	Assign V1 position
MG "Position is", V1= ST	Print Message Stop
EN	End of Program

HINT: ***The accuracy of the MF command is the number of counts that occur in 2 msec. Multiply the speed by 2 msec to obtain the maximum error. MF tests for absolute position. The MF command can also be used when the specified motor is driven independently by an external device.***

MG (Message)

[General]

DESCRIPTION:

The MG command sends data out the specified port. This can be used to alert an operator, send instructions or return a variable value.

ARGUMENTS: MG {Ex} "m", {^n}, V {Fm.n or $m.n} {N} {Sn}where

{Ex}For Ethernet and ‘x’ specifies the Ethernet handle (A,B,C,D,E,F,G,H). Note: if {Ex} is used, it must be the first option after the MG command.

"m" is a text message including letters, numbers, symbols or <ctrl>G. Make sure that maximum line length is not exceeded.

{^n} is an ASCII character specified by the value n in decimal.

V is a variable name or array element where the following specifiers can be used for formatting:

{Fm.n} Display variable in decimal format with m digits to left of decimal, and n to the right.

{$m,n} Display variable in hexadecimal format with m digits to left of decimal, and n to the right.

{N} Suppress carriage return line feed.

{Sn} Display variable as a string of length n where n is 1 thru 6 (n can be omitted, all characters)

***Note Multiple text, variables, and ASCII characters may be used, each must be separated by a comma.**

USAGE:

While Moving	Yes	Default Value	-
In a Program	Yes	Default Format	Variable Format
Command Line	Yes		
Can be Interrogated	No		
Used as an Operand	No		

RELATED COMMANDS:

MG (Message)

EXAMPLES:

Case 1: Message command displays ASCII strings

MG "Good Morning" Displays the string

Case 2: Message command displays variables or arrays

MG "The Answer is", TOTAL {F4.2} Displays the string with the content of variable TOTAL in local format of 4 digits before and 2 digits after the decimal point.

Case 3: Message command sends any ASCII characters to the port.

MG {^13}, {^30}, {^37}, {N} Sends carriage return, characters 0 and 7 followed by no carriage return line feed command to the port.

MO (Motor Off)

[Setting]

DESCRIPTION:

The MO command shuts off the PID control algorithm and the servo enable signal. The controller will continue to monitor the motor position. To turn the motor back on use the SH (Servo Here) command.

The servo cannot be turned off (MO) while it is commanded to move. Issuing the MO command in this mode will cause a command error. Use the ST (Stop), AM (After Motion) or AB (Abort) commands before MO.

ARGUMENTS: *MO XYZW or ABCD*

USAGE:

While Moving	No	Default Value	1
In a Program	Yes	Default Format	1.0
Command Line	Yes		
Can be Interrogated	Yes		
Used as an Operand	Yes		
Virtual Axis	No		

OPERAND USAGE:

_MOn will return the state of the motor where n is an axis letter, 0 = servo loop on and 1 = servo loop off.

RELATED COMMANDS:

SH (Servo Here)

AB (Abort)

AM (After Motion)

ST (Stop)

EXAMPLES:

MO	Turn off motor
SH	Turn motor on
Bob=_MO	Sets Bob equal to the servo status
Bob=	Return value of Bob. If 1, in motor off mode, If 0, in servo mode

MR (Motion Reverse)

[Trippoint]

DESCRIPTION:

The MR command is a trippoint used to control the timing of events. This command will hold up the execution of the following command until the specified motor moves backward and crosses the position specified. The units of the command are in quadrature counts. The MR command can also be used when the encoder is the master and not under servo control.

ARGUMENTS: MR x, y, z, w or MRX=x or MR a, b, c, d where

x, y z, w, or a, b, c, d are signed integers

USAGE:

While Moving	Yes	Minimum Value	-2147483648
In a Program	Yes	Maximum Value	2147483647
Command Line	Yes	Default Value	---
Can be Interrogated	No	Default Format	---
Used as an Operand	No		
Virtual Axis	No		

RELATED COMMANDS:

AR (After Relative)

MF (Motion Forward)

AP (After Absolute Position)

EXAMPLES:

#TEST	Program B
DP0	Define zero
JG 1000	Jog mode (speed of 1000 counts/sec)
BG	Begin move
MR -3000	After passing the position -3000
V1=_TP	Assign current position to variable V1.
MG "Position is", V1	Print Message
ST	Stop
EN	End of Program

HINT: ***MR command accuracy is the number of counts that occur in 2 msec. Multiply speed by 2 msec to obtain maximum error. MR tests for absolute position. The MR command can also be used when the specified motor is driven externally.***

MT (Motor Type)

[Configuration]

DESCRIPTION:

The MT command selects the type of the motor and the polarity of the drive signal. Motor types include standard servo motors which require a voltage in the range of +/- 10 Volts, and step motors, which require pulse and direction signals. The polarity reversal inverts the analog signals for servomotors, and inverts logic level of the pulse train, for step motors.

Warning: This command interacts with the CE command, which reverses the incoming encoder signals. Use caution (motor off, machine estopped) when changing the MT or CE commands. If the two commands are not in agreement with each other, the motor will run away at full speed when enabled.

ARGUMENTS: MT n,n,n,n or MTA=n where

n = 1 Specifies Servo motor

n = -1 Specifies Servo motor with reversed polarity

n = -2 Specifies Step motor with active high step pulses

n = 2 Specifies Step motor with active low step pulses

n = -2.5 Specifies Step motor with reversed direction and active high step pulses

n = 2.5 Specifies Step motor with reversed direction and active low step pulses

USAGE:

While Moving	No	Default Value	1,1,1,1
In a Program	Yes	Default Format	1
Command Line	Yes		
Controller Usage	All controllers		
Virtual Axis	No		

OPERAND USAGE:

_MTn contains the value of the motor type for the specified axis.

RELATED COMMANDS:

CE (Configure Encoder)

EXAMPLES:

MT 1,-1,2,2	Configure a as servo, b as reverse servo, c and d as steppers
MT ?,?	Interrogate motor type
V=_MTA	Assign motor type to variable

MW (Modbus Wait)

[Configuration]

DESCRIPTION:

The MW command sets the controller to wait for the ACK signal from a remote I/O device before going to the next command. With this setting disabled, the controller will continue executing commands after an I/O command that requires it to send a modbus packet. In this mode, the I/O state cannot be guaranteed. Enabling this setting is the default, and is recommended. This is a configuration command and only needs to be set once in the program. This configuration is not burnable, and is set to “enabled” at power up.

ARGUMENTS: MW n where

n is 0 to disable the Modbus Wait function.

n is 1 to enable the Modbus Wait function.

USAGE:

While Moving	Yes	Minimum Value	0
In a Program	Yes	Maximum Value	1
Command Line	Yes	Default Value	0
Can be Interrogated	Yes		
Used as an Operand	Yes		

OPERAND USAGE:

_MW contains the contains current setting of the MW command.

RELATED COMMANDS:

CB (Clear Bit)

MB (Modbus)

SB (Set Bit)

EXAMPLES:

MW1	Enable Modbus Wait function.

Typically this command would be set once at the top of the application program.

~n (Variable Axis Designator)

[Designator]

DESCRIPTION:

The ~N term signifies a varibale axis designator.

ARGUMENTS: ~n=m

n is a **lowercase** letter a though d.

m is a positive integer 0 through 10 where

0 of “X” or “A” (quotes required) = X axis

1 or “Y” or “B” = Y axis

2 or “Z” or “C” = Z axis

3 or “W” or “D” = W axis

8 or “S” = S coordinate system

9 or “T” =T coordinate system

10 or “N” = Virtual N axis

USAGE:

While Moving	Yes	Defualt Value	–
In a Program	Yes	Default Format	1.0
Command Line	Yes		

OPERAND USAGE:

~n contains the axis number 0~10

EXAMPLES:

~a=2;~b=3	Sets ~a to 2 (Z axis). Sets ~b to 3 (W axis)
PR~a=1000	Relative position move 1000 counts on ~a axis (set as Z axis)
JG~b=9000	Set jog speed of ~b axis (set as G axis) to 9000cts/sec
BG~a~b	Begin motion on ~a and ~b axis

Note: This is an axis designator, not a true command.

NB (Notch Bandwidth)

[Tuning]

DESCRIPTION:

The NB command sets real part of the notch poles.

ARGUMENTS: NB *x, y, z, w or NBX=x or NB a, b, c, d where*

x, y z, w, or a, b, c, d are unsigned integers

USAGE:

While Moving	Yes	Minimum value	0
In a Program	Yes	Maximum value	1000000 / 4 * _TM
Command Line	Yes	Default Value	0.5
Used as an Operand	Yes	Default Format	
Can be Interrogated	Yes		

OPERAND USAGE:

_NBn contains the contains the value of the notch bandwidth where n is an axis letter.

RELATED COMMANDS:

NF (Notch Filter)

NZ (Notch Zero)

EXAMPLES:

NBX = 10	Sets the real part of the notch pole to 10 Hz
NOTCH = _NBX	Sets the variable "NOTCH" equal to the notch bandwidth value for the X axis

NF (Notch Filter)

[Tuning]

DESCRIPTION:

The NF command sets the frequency of the notch filter, which is placed in series with the PID compensation.

ARGUMENTS: NF x, y, z, w or NFX=x or NF a, b, c, d where

x, y z, w, or a, b, c, d are unsigned integers

USAGE:

While Moving	Yes	Minimum value	0
In a Program	Yes	Maximum value	1000000 / 4 * _TM
Command Line	Yes	Default Value	0
Can be Interrogated	Yes	Default Format	
Used as an Operand	Yes		

OPERAND USAGE:

_NFn contains the value of notch filter for the specified axis where n is an axis letter.

RELATED COMMANDS:

NB (Notch Bandwidth)

NZ (Notch Zero)

EXAMPLES:

NF, 20	Sets the notch frequency of Y axis to 20 Hz

NO (No Operation)

[General]

DESCRIPTION:

The NO command performs no action in a sequence, but can be used as a comment in a program. After the NO, characters can be given to form a program comment up to the maximum line length. This helps to document a program.

An apostrophe (‘) may also be used instead of the NO to document a program. Comments designated with either the NO or ‘ remain in the program as it is downloaded to the controller, thus occupying some memory space.

ARGUMENTS: NO m where

m is any group of letters, numbers, symbols or <cntrl>G

USAGE:

While Moving	Yes	Default Value	---
In a Program	Yes	Default Format	---
Command Line	Yes		
Can be Interrogated	No		
Used as an Operand	No		

RELATED COMMANDS:

"//" This is a comment command YTerm filters out

"‘" The apostrophe line comment

EXAMPLES:

#A	Program A
NO	No Operation
NO This Program	No Operation
NO Does Absolutely	No Operation
NO Nothing	No Operation
EN	End of Program

NZ (Notch Zero)

[Tuning]

DESCRIPTION:

The NZ command sets the real part of the notch zero.

ARGUMENTS: NZ x, y, z, w or NZX=x or NZ a, b, c, d where

x, y z, w, or a, b, c, d are unsigned integers

USAGE:

While Moving	Yes	Minimum value	0
In a Program	Yes	Maximum value	1000000 / 4 * _TM
Command Line	Yes	Default Value	0.5
		Default Format	

OPERAND USAGE:

_NZn contains the value of the Notch filter zero for the specified axis where n is an axis letter.

RELATED COMMANDS

NB (Notch Bandwidth)

NF (Notch Filter)

EXAMPLES:

NZX = 10	Sets the real part of the notch pole to 10 Hz

OB (Output Bit)

[I/O]

DESCRIPTION:

The OB n, logical expression command defines output bit n = 1 through 8 as either 0 or 1 depending on the result from the logical expression. Any non-zero value of the expression results in a one on the output.

ARGUMENTS: OB n, expression where

n is 1 through 8 denoting output bit

expression is any valid logical expression, variable or array element.

USAGE:

While Moving	Yes	Default Value	---
In a Program	Yes	Default Format	---
Command Line	Yes		
Can be Interrogated	No		
Used as an Operand	No		

EXAMPLES:

OB 1, POS 1	If POS1 is non-zero, Bit 1 is high. If POS1 is zero, Bit 1 is low
OB 2, (@IN[1]&@IN[2])	If Input 1 and Input 2 are both high, then Output 2 is set high
OB 3, COUNT[1]	If the element 1 in the array is zero, clear bit 3, otherwise set bit 3
OB N, COUNT[1]	If element 1 in the array is zero, clear bit N

OE (Off On Error)

[Setting]

DESCRIPTION:

The OE command causes the controller to shut off the motor command if the position error exceeds the limit specified by the ER command or an abort occurs from either the abort input or an AB command.

ARGUMENTS: OE x, y, z, w or OEX=x or OE a, b, c, d where

x, y z, w, or a, b, c, d are 0 or 1

USAGE:

While Moving	Yes	Default Value	0
In a Program	Yes	Default Format	1.0
Command Line	Yes		
Can be Interrogated	Yes		
Used as an Operand	Yes		

OPERAND USAGE:

_OEn contains the status of the off-on-error function where n is an axis letter.

RELATED COMMANDS:

ER (Error Limit)

SH (Servo Here)

#POSERR Error Subroutine

EXAMPLES:

OE 1	Enable OE
OE 0	Disable OE

HINT: The OE command is useful for preventing system damage on excessive error.

OF (Offset)

[Tuning]

DESCRIPTION:

The OF command sets a bias voltage in the motor command output or returns a previously set value. This can be used to counteract gravity or an offset in an amplifier. If the PID values are zero, then the output voltage will be the OF value.

This command is useful when compensating for gravity in a vertical load application.

ARGUMENTS: OF x, y, z, w or OFX=x or OF a, b, c, d where

x, y z, w, or a, b, c, d are signed integers

USAGE:

While Moving	Yes	Minimum Value	-9.9988
In a Program	Yes	Maximum Value	9.9988
Command Line	Yes	Default Value	0
Can be Interrogated	Yes	Default Format	1.4
Used as an Operand	Yes		

OPERAND USAGE:

_OFn contains the offset in volts where n is an axis letter.

EXAMPLES:

OF 1	Set offset to 1 volt
OF ?	Return offset
1.0000	

OP (Output Port)

[I/O]

DESCRIPTION:

The OP command sets 8 bits of data on the output port of the controller simultaneously.

ARGUMENTS: OP m where

m is an integer 0 to 255.

USAGE:

While Moving	Yes	Minimum m Value	0
In a Program	Yes	Maximum m Value	255
Command Line	Yes	Default m Value	0
Can be Interrogated	Yes	Default Format	3.0
Used as an Operand	Yes		

OPERAND USAGE:

_OP contains the status of the outputs.

RELATED COMMANDS:

SB (Set Bit)

CB (Clear Bit)

EXAMPLES:

OP 0	Clear Output Port -- all bits
OP 3	Set outputs 1 and 2; clear the others
OP $FF	Set all outputs to ON.
MG_OP	Message out the status of the outputs
SAA=”MG”,”_OP”	Send command MG_OP to slave controller on handle A
SlaveOut=_SAA	Store the returned value to variable
SAA=”OP”,$OF	Set four outputs ON in slave controller on handle A

@OUT (Output)

[Function]

DESCRIPTION:

@OUT returns the status of the digital output number or variable given in square brackets. Note that the @OUT command is a function, which means that it does not follow the convention of the commands, and does not require the underscore when used as an operand.

ARGUMENTS: @OUT [n] where

n is an integer corresponding to a specific output on the controller. The first output on the controller is denoted as output 1. An SMC–4000 controller has 8 digital outputs plus applicable I/O connected by Modbus.

***Note: When using Modbus devices, the I/O points of the modbus devices are calculated using the following formula:**

n = (SlaveAddress*1000) + (HandleNum*1000) + ((Module-1)*4) + (Bitnum-1)

Slave Address is used when the ModBus device has slave devices connected to it and specified as Addresses 0 to 255.

HandleNum is the handle specifier from A to H (1 - 8).

Module is the position of the module in the rack from 1 to 16.

BitNum is the I/O point in the module from 1 to 4.

USAGE:

While Moving	Yes	Minimum n value	1
In a Program	Yes	Maximum n value	8
Not in a program	Yes	Default Value	---
Can be Interrogated	No	Default Format	---
Used as an Operand	Yes		

EXAMPLES:

#TEST	Program TEST
VAR1=3	Set variable
MG @OUT[VAR1]	Display only the whole number portion of VAR1
EN	End of program

PA (Position Absolute)

[Motion]

DESCRIPTION:

The PA command will set the absolute destination of the next move. The position is referenced to absolute zero. For each single move, the largest position move possible is +/- 2147483647. Units are in quadrature counts.

ARGUMENTS: PA x, y, z, w or PAX=x or PA a, b, c, d where

x, y z, w, or a, b, c, d are signed integers

USAGE:

While Moving	No	Minimum Value	-2147483647
In a Program	Yes	Maximum Value	2147483648
Command Line	Yes	Default Value	---
Can be Interrogated	Yes	Default Format	Position Format
Used as an Operand	Yes		

OPERAND USAGE:

_PAn returns the current commaned position if not moving, and returns the start position if given during motion where n is an axis letter.

RELATED COMMANDS:

PR (Position Relative)

SP (Speed)

AC (Acceleration)

DC (Deceleration)

BG (Begin)

EXAMPLES:

:PA 400	X-axis will go to 400 counts
MG_PAX	
0000000	
:BG	Start the move
:PA 700	X-axis will go to 700 on the next move
:BG	

PF (Position Format)

[Setting]

DESCRIPTION:

The PF command allows the user to format the position numbers such as those returned by TP. The number of digits of integers and the number of digits of fractions can be selected with this command. An extra digit for sign and a digit for decimal point will be added to the total number of digits. If PF is minus, the format will be hexadecimal and a dollar sign will precede the characters. Hex numbers are displayed as 2's complement with the first bit used to signify the sign.

If a number exceeds the format, the number will be displayed as the maximum possible positive or negative number (i.e. 999.99, -999, $8000 or $7FF).

The PF command can be used to format values returned from the following commands:

BL X	PA X
DE X	PR X
DP X	TE
FL X	TP
IP X	

ARGUMENTS: PF m.n where

m is an integer. The negative sign for m specifies hexadecimal representation.

n is an integer

USAGE:

While Moving	Yes	Minimum m Value	-8
In a Program	Yes	Maximum m Value	10
Command Line	Yes	Default m Value	10.0
Can be Interrogated	Yes	Minimum n Value	0
Used as an Operand	Yes	Maximum n Value	4
		Default n Value	0
		Default Format	10.0

OPERAND USAGE:

_PF contains the value of position format parameter.

EXAMPLES:

:TP	Tell position
0000000021	Default format
:PF 5.2	Change format to 5 digits of integers and 2 of fractions
:TP	Tell Position
00021.00	
PF-5.2	New format Change format to hexadecimal*
:TP	Tell Position
$00015.00	Report in hex

PL (Pole)

[Motion]

DESCRIPTION:

The PL command adds a low-pass filter in series with the PID compensation. The digital transfer function of the filter is (I-P)/(Z-P) and the equivalent continuous filter is A/(S+A) where A is the filter cutoff frequency: A=(1/T) In (I/p) rad/sec and T is the sample time.

ARGUMENTS: PL n,n,n,n or PLA=n where

n is a positive number in the range 0 to 0.9999.

USAGE:

While Moving	Yes	Minimum n Value	0
In a Program	Yes	Maximum n Value	0.9999
Command Line	Yes	Default Value	0.0
		Default Format	3.0

OPERAND USAGE:

_PLn contains the value of the pole filter for the specified axis.

RELATED COMMANDS:

KD (Derivative Constant)

KP (Proportional Constant)

KI (Integrator)

NF (Notch Filter)

NZ (Notch Zero)

NB (Notch Bandwidth)

EXAMPLES:

PL .95,.9,.8,.822	Set A-axis Pole to 0.95, B-axis to 0.9, C-axis to 0.8, D-axis pole to 0.822
MG_PLX	Return Pole X only
0.9527	
MG_PLY	Return Pole Y only
0.8997	

PR (Position Relative)

[Motion]

DESCRIPTION:

The PR command sets the incremental distance and direction of the next move. The move is referenced with respect to the current position. The PR command units are in quadrature counts.

ARGUMENTS: PR x, y, z, w or PRX=x or PR a, b, c, d where

x, y z, w, or a, b, c, d are signed integers

USAGE:

While Moving	No	Minimum n Value	-2147483648
In a Program	Yes	Maximum n Value	2147483647
Command Line	Yes	Default Value	0
Can be Interrogated	Yes	Default Format	Position Format setting
Used as an Operand	Yes		
Virtual Axis	Yes		

OPERAND USAGE:

_PRn will return the current incremental distance where n is an axis letter.

RELATED COMMANDS:

BG (Begin)

AC (Acceleration)

DC (Deceleration)

SP (Speed)

IP (Increment Position)

EXAMPLES:

:PR 100	On the next move the X-axis will go 100 counts,
:BG	
:PR ?	Return relative distances
0000000100	

PW (Password)

[Configuration]

DESCRIPTION: PW

The (PW) Password command sets or changes the controller's security password. The command requires two parameters; p,p. Both parameters are the new password up to 8 characters in length. Both parameters must be identical for the new password to be accepted. The password can only be set or changed while the controller is in the "Unlocked" mode, (see the LC command) or a command error will result. Once a valid password is entered, it is automatically burned into the controller EEPROM. The burn causes all parameters listed on the BN page to be discarded.

ARGUMENTS: PW p,pwhere

p,p are identical passwords up to 8 characters in length.

All characters can be alphabetic or numeric.

USAGE:

While Moving	Yes	Default Value	---
In a Program	No	Default Format	---
Command Line	Yes		
Used as an Operand	No		
Can be Interrogated	No		

RELATED COMMANDS:

LC (Lock Controller)

EXAMPLES:

PW MOTION,MOTION	Set a new password "MOTION"
LC MOTION,1	Lock Controller
LC MOTION,0	Unlock Controller

QA (Query Auxilliary Encoder Unmodularized Position)

[Motion]

DESCRIPTION:

Similar to the TD command, this command returns the current position of the auxilliary encoder(s) for the ECAM axis. Unlike TD, the true encoder position is returned rather than the position modulus in the CAM cycle. If QA is called on an axis that is not in ECAM mode, QA will behave identically to TD.

ARGUMENTS: QA nnnn where

n is X, Y, Z, W, A, B, C, D or any combination specifying the axis or axes.

USAGE:

While Moving	Yes	Default Value	0
In a Program	Yes	Default Format	PF
Command Line	Yes		

OPERAND USAGE:

_QAn contains the current position value for the specified auxilliary encoder.

RELATED COMMANDS:

TD (Tell Dual (Auxiliary) Encoder)

PF (Position Format)

EXAMPLES:

The A-axis auxilliary encoder is at position 200, the B-axis is at -10, the C-axis is at 0, and the D-axis is at -110. The returned parameter units are in quadrature counts.

:PF 7	Position format of 7
:QA 0000200, -0000010, 0000000, -0000110	Return A, B, C, D main encoder positions
:QAA 0000200	Return the A main encoder position
:QAB -0000010	Return the B main encoder position
:PF-6.0	Change to hex format
:QA $0000C8, $FFFFF6, $000000, $FFFF93	Retrun A, B, C, D in hex
: Position=_QAA	Assign the variable, position, the value of QAA

QD (Download Array)

[General]

DESCRIPTION:

The QD command transfers array data from the host computer to the SMC–4000. QD array[ ],start,end requires that the array name be specified along with the first element of the array and last element of the array. The array elements can be separated by a comma (,) or by <CR><LF>. The downloaded array is terminated by a <control>Z, <control>Q, <control>D or \.

ARGUMENTS: QD array[ ], start, end where

"array[ ]" is a valid array name

"start" is the first element of the array (default=0)

"end" is the last element of the array (default=last element)

USAGE:

While Moving	No	Default Value	0
In a Program	Yes	Default Format	Position Format
Command Line	Yes		
Can be Interrogated	No		
Used as an Operand	No		

RELATED COMMANDS:

UL (Upload)

QP (Query Unmodularized Position)

[Motion]

DESCRIPTION:

Similar to the TP command, this command returns the current position of the main encoder(s) for the ECAM axis. Unlike TD, the true encoder position is returned rather than the position modulus in the CAM cycle. If QA is called on an axis that is not in ECAM mode, QP will behave identically to TD.

ARGUMENTS: QP nnnn where

n is X, Y, Z, W, A, B, C, D or any combination specifying the axis or axes.

USAGE:

While Moving	Yes	Default Value	0
In a Program	Yes	Default Format	PF
Command Line	Yes		

OPERAND USAGE:

_QPn contains the current position value for the specified main encoder.

RELATED COMMANDS:

TD (Tell Dual (Auxiliary) Encoder)

PF (Position Format)

EXAMPLES:

The A-axis main encoder is at position 200, the B-axis is at -10, the C-axis is at 0, and the D-axis is at -110. The returned parameter units are in quadrature counts.

:PF 7	Position format of 7
:QP 0000200, -0000010, 0000000, -0000110	Return A, B, C, D main encoder positions
:QPA 0000200	Return the A main encoder position
:QPB -0000010	Return the B main encoder position
:PF-6.0	Change to hex format
:QP $0000C8, $FFFFF6, $000000, $FFFF93	Retrun A, B, C, D in hex
: Position=_QPA	Assign the variable, position, the value of QAA

QR (Data Record)

[General]

DESCRIPTION:

The QR command causes the controller to return a record of information regarding controller status. This status information includes 4 bytes of header information and specific blocks of information as specified by the command arguments. The details of the status information is described in Chapter 3 of this user's manual.

This command, along with the QZ (Return Data Record Information) command can be very useful for accessing complete controller status. The QR command will return 4 bytes of header information and specific blocks of information as specified by the command arguments: QR ABCDSTI

ARGUMENTS: QR nnnn where

n is A,B,C,D,S,T, or I or any combination to specify the axis, axes, sequence, or I/O status

S and T represent the S and T coordinated motion planes

I represents the status of the I/O

USAGE:

While Moving	Yes	Default Value	---
In a Program	Yes	Default Format	---
Command Line	Yes		
Controller Usage	No		

RELATED COMMANDS:

QZ (Return Data Record Information)

HINT: The reults of the QR command are in binary format.

Each argument corresponds to a block of information according to the Data Record Map below. If no argument is given, the entire data record map will be returned. Note that the data record size will depend on the number of axes.

Data Record Map

This command, along with the QZ command can be very useful for accessing complete controller status.

The QR command will return 4 bytes of header information and specific blocks of information as specified by the command arguments: QR ABCDEFGHST

Each argument corresponds to a block of information according to the Data Record Map below. If no argument is given, the entire data record map will be returned. Note that the data record size will depend on the number of axes.

DATA TYPE	ITEM	BLOCK
UB	1st byte of header	Header
UB	2nd byte of header	Header
UB	3rd byte of header	Header
UB	4rth byte of header	Header
UW	sample number	I block
UW	general input 0	I block

DATA TYPE	ITEM	BLOCK
UW	general input 1	I block
UW	general input 2	I block
UW	general input 3	I block
UW	general input 4	I block
UW	general input 5	I block
UW	general input 6	I block
UW	general input 7	I block
UW	general input 8	I block
UW	general input 9	I block
UW	general output 0	I block
UW	general output 1	I block
UW	general output 2	I block
UW	general output 3	I block
UW	general output 4	I block
UW	general output 5	I block
UW	general output 6	I block
UW	general output 7	I block
UW	general output 8	I block
UW	general output 9	I block
UW	error code	I block
UW	general status	I block
UW	segment count of coordinated move for S plane	S block
UW	coordinated move status for S plane	S block
UW	distance traveled in coordinated move for S plane	S block
UW	segment count of coordinated move for T plane	T block
UW	coordinated move status for T plane	T block
UW	distance traveled in coordinated move for T plane	T block
UW	a axis status	A block
UW	a axis switches	A block
UW	a axis stopcode	A block
UW	a axis reference position	A block
UW	a axis motor position	A block
UW	a axis position error	A block
UW	a axis auxiliary position	A block
UW	a axis velocity	A block
UW	a axis torque	A block
UW	b axis status	B block
UW	b axis switches	B block
UW	b axis stopcode	B block
UW	b axis reference position	B block
UW	b axis motor position	B block

DATA TYPE	ITEM	BLOCK
UW	b axis position error	B block
UW	b axis auxiliary position	B block
UW	b axis velocity	B block
UW	b axis torque	B block
UW	c axis status	C block
UW	c axis switches	C block
UW	c axis stopcode	C block
UW	c axis reference position	C block
UW	c axis motor position	C block
UW	c axis position error	C block
UW	c axis auxiliary position	C block
UW	c axis velocity	C block
UW	c axis torque	C block
UW	d axis status	D block
UW	d axis switches	D block
UW	d axis stopcode	D block
UW	d axis reference position	D block
UW	d axis motor position	D block
UW	d axis position error	D block
UW	d axis auxiliary position	D block
UW	d axis velocity	D block
UW	d axis torque	D block
UW	e axis status	E block
UW	e axis switches	E block
UW	e axis stopcode	E block
UW	e axis reference position	E block
UW	e axis motor position	E block
UW	e axis position error	E block
UW	e axis auxiliary position	E block
UW	e axis velocity	E block
UW	e axis torque	E block
UW	f axis status	F block
UW	f axis switches	F block
UW	f axis stopcode	F block
UW	f axis reference position	F block
UW	f axis motor position	F block
UW	f axis position error	F block
UW	f axis auxiliary position	F block
UW	f axis velocity	F block
UW	f axis torque	F block
UW	g axis status	G block
UW	g axis switches	G block
UW	g axis stopcode	G block

DATA TYPE	ITEM	BLOCK
UW	g axis reference position	G block
UW	g axis motor position	G block
UW	g axis position error	G block
UW	g axis auxiliary position	G block
UW	g axis velocity	G block
UW	g axis torque	G block
UW	h axis status	H block
UW	h axis switches	H block
UW	h axis stopcode	H block
UW	h axis reference position	H block
UW	h axis motor position	H block
UW	h axis position error	H block
UW	h axis auxiliary position	H block
UW	h axis velocity	H block

Note: UB = Unsigned Byte, UW = Unsigned Word, SW = Signed Word, SL = Signed Long Word

Explanation of Status Information and Axis Switch Information

Header Information - Byte 0, 1 of Header:

BIT 15	BIT 14	BIT 13	BIT 12	BIT 11	BIT 10	BIT 9	BIT 8
1	N/A	N/A	N/A	N/A	I Block Present in Data Record	T Block Present in Data Record	S Block Present in Data Record
BIT 7	**BIT 6**	**BIT 5**	**BIT 4**	**BIT 3**	**BIT 2**	**BIT 1**	**BIT 0**
H Block Present in Data Record	G Block Present in Data Record	F Block Present in Data Record	E Block Present in Data Record	D Block Present in Data Record	C Block Present in Data Record	A Block Present in Data Record	B Block Present in Data Record

Bytes 2, 3 of Header:

Bytes 2 and 3 make a word which represents the Number of bytes in the data record, including the header. Byte 2 is the low byte and byte 3 is the high byte

Note: The header information of the data records is formatted in little endian.

General Status Information (1 Byte)

BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0
Program Running	N/A	N/A	N/A	N/A	Waiting for input from IN (Input Variable) command	Trace On	Echo On

Axis Switch Information (1 Byte)

BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0
Latch Occurred	State of Latch Input	N/A	N/A	State of Forward Limit	State of Reverse Limit	State of Home Input	SM Jumper Installed

Axis Status Information (2 Byte)

BIT 15	BIT 14	BIT 13	BIT 12	BIT 11	BIT 10	BIT 9	BIT 8
Move in Progress	Mode of Motion PA (Position Absolute) or PR (Position Relative)	Mode of Motion PA (Position Absolute) only	FE (Find Edge) in Progress	HM (Home) in Progress	1st Phase of HM (Home) complete	2nd Phase of HM (Home) complete or FI (Find Index) command issued	Mode of Motion Coord. Motion
BIT 7	**BIT 6**	**BIT 5**	**BIT 4**	**BIT 3**	**BIT 2**	**BIT 1**	**BIT 0**
Negative Direction Move	Mode of Motion Contour	Motion is slewing	Motion is stopping due to ST (Stop) or Limit Switch	Motion is making final decel.	Latch is armed	Off-On-Error occurred	Motor Off

Coordinated Motion Status Information for S or T plane (2 Byte)

BIT 15	BIT 14	BIT 13	BIT 12	BIT 11	BIT 10	BIT 9	BIT 8
Move in Progress	N/A	N/A	N/A	N/A	N/A	N/A	N/A
BIT 7	**BIT 6**	**BIT 5**	**BIT 4**	**BIT 3**	**BIT 2**	**BIT 1**	**BIT 0**
N/A	N/A	Motion is slewing	Motion is stopping due to ST (Stop) or Limit Switch	Motion is making final decel.	N/A	N/A	N/A

Notes Regarding Velocity and Torque Information

The velocity information that is returned in the data record is 64 times larger than the value returned when using the TV (Tell Velocity) command. See command reference for more information about TV. The Torque information is represented as a number in the range of +/-32767. Maximum negative torque is -32767. Maximum positive torque is 32767. Zero torque is 0.

QZ (Return Data Record Information) Command

The QZ command can be very useful when using the QR command, since it provides information about the controller and the data record. The QZ command returns the following 4 bytes of information.

BYTE #	INFORMATION
0	Number of axes present
1	number of bytes in general block of data record
2	number of bytes in coordinate plane block of data record
3	Number of Bytes in each axis block of data record

QU (Upload Array)

[General]

DESCRIPTION:

The QU command transfers array data from the SMC–4000 to a host computer. QU requires that the array name be specified along with the first element of the array and last element of the array. The uploaded array will be followed by a <control>Z as an end of text marker.

ARGUMENTS: QU array[], start, end, delim where

"array[ ]" is a valid array name

"start" is the first element of the array (default=0)

"end" is the last element of the array (default=last element)

"delim" specifies the character used to delimit the array elements. If delim is 1, then the array elements will be separated by a comma. Otherwise, the elements will be separated by a carriage return.

USAGE:

While Moving	No	Default Value	0
In a Program	Yes	Default Format	Position Format
Command Line	Yes		
Can be Interrogated	No		
Used as an Operand	No		

RELATED COMMANDS:

DL (Download)

QY (Query Yaskawa Absolute Encoder Alarm)

[General]

DESCRIPTION:

The Query Yaskawa Absolute Encoder Alarm (QY) command displays the serial data received from an absolute encoder when th eposition was requested usign the AE (Absolute Encoder) command. Usually, the data inthe QY command is the number of revolutions of the servo from the absolute zero point. If there was an encoder alarm, one of the following codes will be stored in the QY command.

ALMRMOA= Backup Alarm

ALMRMOB= Checksum error

ALMRMOD=Battery Alarm

ALMRMOE=Battery/Backup Combination Error

ALMRMOH=Absolute Error

ALMRMOP=Overspeed

ARGUMENTS: None

USAGE:

While Moving	No	Default Value	---
In a Program	Yes	Default Format	---
Command Line	Yes		
Can be Interrogated	Yes		
Used as an Operand	No		

RELATED COMMANDS:

AE (Absolute Encoder)

TY (Tell Yaskawa Absolute Encoder)

QZ (Return Data Record Information)

[General]

DESCRIPTION:

The QZ command is an interrogation command that returns information regarding the Data Record (QR). The controller's response to this command will be the return of 4 integers separated by commas. The four fields represent the following:

First field returns the number of axes.

Second field returns the number of bytes to be transferred for general status

Third field returns the number bytes to be transferred for coordinated move status

Fourth field returns the number of bytes to be transferred for axis specific information

ARGUMENTS: QZ

USAGE:

While Moving	Yes	Default Value	---
In a Program	Yes	Default Format	
Command Line	Yes		

RELATED COMMANDS:

QR (Data Record)

RA (Record Array)

[General]

DESCRIPTION:

The RA command selects up to eight arrays for automatic data capture. The selected arrays must have been dimensioned by the DM command. The data to be captured is specified by the RD command and time interval by the RC command.

ARGUMENTS: RA n [ ],m [ ],o [ ],p [ ]. q[ ], r[ ], s[ ], t[ ] where

n,m,o,p,q,r,s,t are dimensioned arrays as defined by DM command. The [ ] contain nothing.

USAGE:

While Moving	Yes	Default Value	---
In a Program	Yes	Default Format	---
Command Line	Yes		
Can be Interrogated	No		
Used as an Operand	No		

RELATED COMMANDS:

DM (Dimension Array)

RD (Record Data)

RC (Record)

EXAMPLES:

#Record	Label
DM POS[100], ERR[100]	Define array
RA POS[], ERR[ ]	Specify Record Mode
RD _TPX	Specify data type for record
RC 1	Begin recording at 2 msec intervals
PR 1000;BG	Start motion
EN	End

HINT: ***The record array mode is useful for recording the real-time motor position during motion. The data is automatically captured in the background and does not interrupt the program sequencer. The record mode can also be used for a teach or learn of a motion path.***

RC (Record)

[General]

DESCRIPTION:

The RC command begins recording for the RA (Record Array) Mode. RC 0 stops recording.

ARGUMENTS: RC n,m where

n is an integer 1 thru 8 and specifies 2^n samples between records. RC 0 stops recording.

m is optional and specifies the number of records to be recorded. If m is not specified, the DM (Dimension Array) number will be used. A negative number for m causes circular recording over array addresses 0 to m-1. The address for the array element for the next recording can be interrogated with _RD (Record Data).

USAGE:

While Moving	Yes	Minimum n Value	0
In a Program	Yes	Maximum n Value	8
Command Line	Yes	Default n Value	---
Can be Interrogated	Yes	Minimum m Value	-1
Used as an Operand	Yes	Maximum m Value	8000
		Default m Value	---
		Default Format	---

OPERAND USAGE:

_RC contains status of recording '1' if recording, '0' if not recording.

RELATED COMMANDS:

DM (Dimension Array)

RD (Record Data)

RA (Record Array)

EXAMPLES:

#RECORD	Record
DM Torque[1000]	Define Array
RA Torque[]	Specify Record Mode
RD _TT	Specify Data Type
RC 2	Begin recording, set 4 servo samples between records
JG 1000;BG	Begin motion
#A;JP #A,_RC=1	Loop until done
MG "DONE RECORDING"	Print message
EN	End program

RD (Record Data)

[General]

DESCRIPTION:

The RD command specifies the data type to be captured for the RA (Record Array) mode. The data types include:

Data Type	Meaning
_DE	2nd encoder
_TP	Position
_TE	Position error
_SH	Commanded position
_RL	Latched position
_TI	Inputs
_OP	Outputs
_TS	Switches, only 0-4 bits valid
_SC	Stop code
_TT	Tell torque

ARGUMENTS: RD m1, m2, m3, m4, m5, m6, m7, m8 where

the arguments are the data type to be captured using the record array feature. The order is important. Each of the eight data types corresponds with the array specified in the RA command.

USAGE:

While Moving	Yes	Default Value	---
In a Program	Yes	Default Format	---
Command Line	Yes		
Can be Interrogated	No		
Used as an Operand	Yes		

OPERAND USAGE:

_RD contains the address for the next array element for recording.

RELATED COMMANDS:

RA (Record Array)

RC (Record)

DM (Dimension Array)

EXAMPLES:

DM ERRORX[50]	Define array
RA ERRORX[]	Specify record mode
RD _TE	Specify data type
RC1	Begin record
JG 1000;BG	Begin motion

RE (Return from Error)

[Program Flow]

DESCRIPTION:

The RE command is used to end a position error handling subroutine or limit switch handling subroutine. The error handling subroutine begins with the #POSERR label. The limit switch handling subroutine begins with the #LIMSWI. An RE at the end of these routines causes a return to the main program. Care should be taken to be sure the error or limit switch conditions no longer occur to avoid re-entering the subroutines. If the program sequencer was waiting for a trippoint to occur, prior to the error interrupt, the trippoint condition is preserved on the return to the program if RE1 is used. RE0 clears the trippoint. To avoid returning to the main program on an interrupt, use the ZS (Zero Subroutine Stack) command to zero the subroutine stack. No RE is needed after ZS (Zero Subroutine Stack). After using ZS, use a JP (Jump to Program Location) command to return to a key location in the main program.

ARGUMENTS: RE n where

0 or nothing clears the interrupted trippoint

1 restores state of trippoint

USAGE:

While Moving	No	Minimum n Value	0
In a Program	Yes	Maximum n Value	1
Command Line	No	Default Value	0
Can be Interrogated	No	Default Format	---
Used as an Operand	No		

RELATED COMMANDS:

#POSERR	Excessive Position Error Special Label
#LIMSWI	Limit Switch Special Label

EXAMPLES:

#A;JP #A;EN	Label for main program
#POSERR	Begin Error Handling Subroutine
MG "ERROR"	Print message
SB1	Set output bit 1
RE	Return to main program and clear trippoint

***Note An application program must be executing for the #LIMSWI and #POSERR subroutines to function.**

RI (Return from Interrupt)

[Program Flow]

DESCRIPTION:

The RI command is used to end the interrupt subroutine beginning with the label #ININT. An RI (Return from Interrupt) at the end of this routine causes a return to the main program. The RI command also re-enables input interrupts. If the program sequencer was interrupted while waiting for a trippoint, such as WT (Wait), RI1 restores the trippoint upon return to the program. RI0 clears a trippoint. To avoid returning after an interrupt, use the ZS (Zero Subroutine Stack) command to zero the subroutine stack. Check the example section for more details about using interrupts.

ARGUMENTS: RI n where

n = 0 or 1

0 or nothing clears interrupt trippoint

1 restores trippoint

USAGE:

While Moving	No	Minimum n Value	0
In a Program	Yes	Maximum n Value	1
Command Line	Yes	Default Value	---
Can be Interrogated	No	Default Format	---
Used as an Operand	No		

RELATED COMMANDS:

#ININT Input interrupt subroutine

II (Input Interrupt)

EXAMPLES:

#A;II1;JP #A;EN	Program label
#ININT	Begin interrupt subroutine
MG "INPUT INTERRUPT"	Print Message
SB 1	Set output line 1
RI 1	Return to the main program and restore trippoint

***Note An applications program must be executing for the #ININT subroutine to function.**

RL (Report Latch)

[General]

DESCRIPTION:

The RL command will return the last position captured by the latch. The latch must first be armed by the AL (Arm Latch) command. The armed state of the latch can be configured using the CN (Configure Limit Switches) command.

ARGUMENTS: RLn where

n = XYZW or ABCD for the main encoder latch.

USAGE:

While Moving	Yes	Default Value	n/a
In a Program	Yes	Default Format	Position Format
Command Line	Yes		
Can be Interrogated	No		
Used as an Operand	Yes		

OPERAND USAGE:

_RLn contains the main encoder latched position where n is an axis letter.

RELATED COMMAND:

AL (Arm Latch)

CN (Configure Limit Switches)

LT (Latch Target)

EXAMPLES:

JG 5000	Set up to jog
BG	Begin jog
AL	Arm the latch; assume that after about 2 seconds, input goes low
#WAIT; JP #WAIT,_ALX=1	Wait here while latch is still armed
RL	Report the latch
10000	

@RND (Round)

[Function]

DESCRIPTION:

@RND rounds a number or variable given in square brackets. Note that the @RND command is a function, which means that it does not follow the convention of the commands, and does not require the underscore when used as an operand.

ARGUMENTS: @RND [n] ***where***

n is a number

USAGE:

While Moving	Yes	Minimum n value	-2147483648.9999
In a Program	Yes	Maximum n value	2147483648.9999
Not in a program	Yes	Default Value	---
Can be Interrogated	No	Default Format	---
Used as an Operand	Yes		

EXAMPLES:

#TEST	Program TEST
VAR1=123.456	Set variable
MG @RND[VAR1]	Display the value of VAR1 rounded to the nearest integer
VAR2=@RND[VAR1]+25	Perform calculation
EN	End of program

RP (Reference Position)

[Motion]

DESCRIPTION:

The RP command will return the commanded position of the servo. This is updated every sample period by the profiler. RP-TP=TE. The units are in quadrature counts.

ARGUMENTS: RPn where

n = XYZW or ABCD or N

USAGE:

While Moving	Yes	Default Value	n/a
In a Program	Yes	Default Format	Position Format
Command Line	Yes		
Can be Interrogated	Yes		
Used as an Operand	Yes		
Virtual Axis	Yes		

OPERAND USAGE:

_RPn contains the commanded position where n is an axis letter.

RELATED COMMANDS:

TP (Tell Position)

EXAMPLES:

PR 10000	Position Relative move
BG	Begin motion
AM	After Motion
RP	Display the Reference Position
10000	

RS (Reset)

[General]

DESCRIPTION:

The RS command resets the processor to its power-on condition. The previously saved (burned) state of the controller, along with parameter values, and saved sequences are restored.

ARGUMENTS: None

USAGE:

While Moving	Yes	Default Value	0
In a Program	No	Default Format	---
Command Line	Yes		
Can be Interrogated	No		
Used as an Operand	No		

EXAMPLES:

RS	Reset the controller

<control>R<control>S (Master Reset)

[General]

DESCRIPTION:

The Master Reset command resets the SMC–4000 to factory default settings and erases the application program. <control>R<control>S are the ASCII characters 18 and 19.

USAGE:

While Moving	Yes	Default Value	---
In a Program	No	Default Format	---
Command Line	Yes		
Can be Interrogated	No		
Used as an Operand	No		

RU (Unmodularized Latch Position)

[Position]

DESCRIPTION:

Similar to RL (Report Latch), RU retuns the umodularized latched encoder position for an ECAM axis. Unlike RL, the true encoder position is returned rather than the position modulus of the cam cycle. If RU is called on an axis that is not in ECAM mode, Ru will behave identically to RL.

ARGUMENTS: RU nnnn where

n is X, Y, Z, W, A, B, C, D or any combination specifying the axis or axes.

USAGE:

While Moving	No	Default Value	0
In a Program	Yes	Default Format	PF
Command Line	Yes		

OPERAND USAGE:

_RUn contains the unmodularized latched encoder position of the specified axis.

RELATED COMMANDS:

AL (Arm Latch)

RL (Report Latch)

EXAMPLES:

:AL	Arm latch and assume that input goes low after 2 seconds.
:RU 1000	Report the latch

<control>R<control>U (Firmware Revision)

[General]

DESCRIPTION:

The Revision command causes the controller to return the firmware revision information.

USAGE:

While Moving	Yes	Default Value	-
In a Program	No	Default Format	-
Command Line	Yes		
Can be Interrogated	No		
Used as an Operand	No		

SA (Send Command)

[General]

DESCRIPTION:

SA sends a command from one controller to another in a peer to peer configuration. Check IHn4 to be sure that the command was accepted by the peer controller. Use the MG (Message) command to send information to the other devices.

ARGUMENTS: SAh=arg or SAh= arg, arg, arg, arg, arg, arg, arg, arg where

h is the handle being used to send commands to the slave controller.

arg is a number, controller operand, variable, mathematical function, or string; the range for numeric values is 4 bytes of integer (2^{31}) followed by two bytes of fraction (+/- 2,147,483,647.9999). The maximum number of characters for a string is 6. Strings are identified by quotations.

Typical usage would have the first argument as a string such as KI (Integrator) and the subsequent arguments as the arguments to the command: Example SAF= “KI”,2 would send the command KI2 to the slave controller on handle F.

USAGE:

While Moving	Yes	Default Value	---
In a Program	Yes	Default Format	---
Command Line	Yes		

OPERAND USAGE:

_SAhn gives the value of the response to the command sent with an SA command. The h value represents the handle A thru H and the n value represents the specific field returned from the controller (1-8). If the specific field is not used, the operand will be (-2^{31}).

RELATED COMMANDS:

IH (Internet Handle)

EXAMPLES:

SAA=”KI”,2	Sends the command to Handle A (slave controller): KI 2
SAA=”TE”	Sends the command to Handle A (slave controller): TE (Tell Error)
MG_SAA : 132	Display the content of the operand _SAA (first response to TE command

SB (Set Bit)

[I/O]

DESCRIPTION:

The SB command sets one of eight bits on the output port, or Modbus I/O.

ARGUMENTS: SB n where

n is an integer in the range 1 to 8 decimal or Modbus address.

MODBUS:

***Note When using Modbus devices, the I/O points of the modbus devices are calculated using the following formula:**

n = (SlaveAddress*1000) + (HandleNum*1000) + ((Module-1)*4) + (Bitnum-1)

Slave Address is used when the ModBus device has slave devices connected to it and specified as Addresses 0 to 255. Please note that the use of slave devices for modbus are very rare and this number will usually be 0.

HandleNum is the handle specifier from A to H (1 - 8).

Module is the position of the module in the rack from 1 to 16.

BitNum is the I/O point in the module from 1 to 4.

USAGE:

While Moving	Yes	Default Value	---
In a Program	Yes	Default Format	---
Command Line	Yes		
Can be Interrogated	No		
Used as an Operand	No		

RELATED COMMAND:

CB (Clear Bit)

OB (Output Bit)

OP (Output Port)

@OUT (Output)

EXAMPLES:

SB 3	Set output line 3
SB 1	Set output line 1
SB 6002	Set output 2 on Modbus device on handle F

SC (Stop Code)

[Status]

DESCRIPTION:

The SC command allows the user to determine why a motor stops. The controller responds with the stop code as follows:

CODE	MEANING	CODE	MEANING
0	Motors are running, independent mode	11	Stopped by selective Abort Input
1	Motors stopped at commanded independent position	40	Topped by latch target
2	Decelerating or stopped by FWD limit switch or software limit, LT (Latch Target)	41	Stopped due to limit switch or stop command in LT (Latch Target) mode
3	Decelerating or stopped by REV limit switch or software limit, BL (Backward Limit)	42	Stopped short of LT (Latch Target) distance
4	Decelerating or stopped by ST (Stop) command	50	Contour running
6	Stopped by Abort input	51	Contour Stop
7	Stopped byAB (Abort) command	99	MC (Motion Complete) timeout
8	Decelerating or stopped by Off-on-Error (OE1)	100	Motors are running, vector sequence
9	Stopped after FE (Find Edge)	101	Motors stopped at commanded vector
10	Stopped after HM (Home)		

ARGUMENTS: SC XYZW or ABCD

USAGE:

While Moving	Yes	Default Value	---
In a Program	Yes	Default Format	3.0
Command Line	Yes		
Can be Interrogated	No		
Used as an Operand	Yes		
Virtual Axis	Yes		

OPERAND USAGE:

_SCn contains the value of the stop code where n is an axis letter.

EXAMPLES:

Tom=_SCX	Assign the Stop Code to variable Tom

SH (Servo Here)

[General]

DESCRIPTION:

The SH command tells the controller to use the current motor position as the commanded position and to enable servo control here. PID control starts when this command is issued.

The SH command should not be used multiple times in a program loop, for the axis may appear to drift especially if doing relative moves.

ARGUMENTS: SH XYZW or ABCD

USAGE:

While Moving	No	Default Value	---
In a Program	Yes	Default Format	---
Command Line	Yes		
Can be Interrogated	No		
Used as an Operand	No		

RELATED COMMANDS:

MO (Motor Off)

EXAMPLES:

SH	Servo motor

@SIN (Sine)

[Function]

DESCRIPTION:

@SIN returns the sin of a number or variable given in square brackets using units of degrees. Note that the @SIN command is a function, which means that it does not follow the convention of the commands, and does not require the underscore when used as an operand.

ARGUMENTS: @SIN [n] ***where***

n is a number

USAGE:

While Moving	Yes	Minimum n value	-32768
In a Program	Yes	Maximum n value	32768
Not in a program	Yes	Default Value	---
Can be Interrogated	No	Default Format	---
Used as an Operand	Yes		

EXAMPLES:

#TEST	Program TEST
VAR1=60	Set variable
MG @SIN[VAR1]	Display the value of the sine of VAR1
VAR2=@SIN[VAR1]+9	Perform calculation
EN	End of program

OPERAND USAGE:

SL (Single Step)

[Function]

DESCRIPTION:

For debugging purposes. Single Step through the program after execution has paused at a BK (Breakpoint). Optional arguments allow the user to specify the number of lines to execute before pausing again. The BK command resumes normal program execution.

ARGUMENTS: SL n ***where***

n is an integer representing the number of lines to execute before pausing again.

USAGE:

While Moving	Yes	Default Value	1
In a Program	No		
Not in a program	Yes		

RELATED COMMANDS:

BK (Breakpoint)

TR (Trace Mode)

EXAMPLES:

BK 3	Pause at lime 3 (the 4th line) in thread 0
BK5	Continue to line 5
SL	Execute the next line
SL 3	Execute the next 3 lines
BK	Resume normal execution

SP (Speed)

[Motion]

DESCRIPTION:

This command sets the slew speed for independent moves. The parameters input will be rounded down to the nearest factor of 2 and the units of the parameter are in counts per second.

Speed can be changed on the fly and the controller will accel/decel at the current settings to the new speed.

***Note Negative values will be interpreted as the absolute value.**

ARGUMENTS: SP x, y, z, w or SPX=x or SP a, b, c, d where

x, y z, w, or a, b, c, d are unsigned integers

USAGE:

While Moving	Yes	Minimum n Value	0
In a Program	Yes	Maximum n Value	12,000,000
Command Line	Yes	Default Value	25000
Can be Interrogated	Yes	Default Format	Position Format
Used as an Operand	Yes		
Virtual Axis	Yes		

OPERAND USAGE:

_SPn contains the current speed setting where n is an axis letter.

RELATED COMMANDS:

AC (Acceleration)

DC (Deceleration)

PR (Position Relative)

BG (Begin)

EXAMPLES:

PR 2000	Specify position relative move
SP 5000	Specify speeds
BG	Begin motion of all axes
AM	After motion is complete

***Note SP is not a "mode" of motion like JOG (JG).**

@SQR (Square Root)

[Function]

DESCRIPTION:

@SQR returns the square root of a number or variable given in square brackets. Note that the @SQR command is a function, which means that it does not follow the convention of the commands, and does not require the underscore when used as an operand. This function will treat negative numbers as positive numbers.

ARGUMENTS: @SIN [n] ***where***

n is a number

USAGE:

While Moving	Yes	Minimum n value	0
In a Program	Yes	Maximum n value	2147483647.9999
Not in a program	Yes	Default Value	---
Can be Interrogated	No	Default Format	---
Used as an Operand	Yes		

EXAMPLES:

#TEST	Program TEST
VAR1=60	Set variable
MG @SQR[VAR1]	Display the value of the sine of VAR1
VAR2=@SQR[VAR1]+9	Perform calculation
EN	End of program

ST (Stop)

[Motion]

DESCRIPTION:

The ST command stops commanded motion. The motor will come to a decelerated stop.

ST is used by itself from any port (not with a program).

ARGUMENTS: ST XYZWST or ABCD where

XYZW or ABCD are axis designators. S or T indicates an interpolation sequence. No argument specifies that motion on all axes will be stopped. Note that the profiler will continue to generate new points so that the axes will come to a decelerated stop.

USAGE:

While Moving	Yes	Default Value	---
In a Program	Yes	Default Format	---
Command Line	Yes		
Can be Interrogated	No		
Used as an Operand	No		

RELATED COMMANDS:

BG (Begin)

MC (Motion Complete)

DC (Deceleration)

EXAMPLES:

ST	Stop motion

HINT: Use the AM (After Motion) command, to wait for motion to be stopped.

@TAN (Tangent)

[Function]

DESCRIPTION:

@TAN returns the tangent of a number or variable given in square brackets using units of degrees. Note that the @TAN command is a function, which means that it does not follow the conventions of the commands, and does not require an underscore when used as an operand.

ARGUMENTS: @TAN [n] where n is a number

USAGE:

While Moving	Yes	Minimum *n* Value	-2147483647
In a Program	Yes	Maximum *n* Vavue	2147483674
Not in a Program	Yes	Default Value	---
Can be Interrogated	No	Default Format	---
Used as an Operand	Yes		

RELATED COMMANDS:

@SIN (Sine)

@COS (Cosine)

EXAMPLES:

#TEST	Program TEST
VAR1=60	Set Variable
MG@TAN[VAR1]	Display the value of the tangent of VAR1
VAR2=@TAN[VAR1]+9	Perform Calculation
EN	End of Program

TB (Tell Status Byte)

[Status]

DESCRIPTION:

The TB command returns status information from the controller as a decimal number. Each bit of the status byte denotes the following condition when the bit is set (high):

BIT	STATUS
Bit 7	Executing program
Bit 6	N/A
Bit 5	Contouring
Bit 4	Executing error or limit switch routine
Bit 3	Input interrupt enabled
Bit 2	Executing input interrupt routine
Bit 1	N/A
Bit 0	Echo on

ARGUMENTS: None

USAGE:

While Moving	Yes	Default Value	---
In a Program	Yes	Default Format	1.0
Command Line	Yes		
Can be Interrogated	Yes		
Used as an Operand	Yes		

OPERAND USAGE:

_TB contains the status byte.

EXAMPLES:

TB	Tell status information from the controller
65	Executing program and echo on ($2^6 + 2^0 = 64 + 1 = 65$)

TC (Tell Code)

[Status]

DESCRIPTION:

The TC command returns a number between 1 and 255. This number is a code that reflects why a command was not accepted by the controller. This command is useful when the controller halts execution of a program at a command or when the response to a command is a question mark. Entering the TC command will provide the user with a code as to the reason. After TC has been read, it is set to zero. TC 1 returns the text message as well as the numeric code.

***Note _ED returns the line number that last had an error.**

ARGUMENTS: TC n

n=0 returns code only

n=1 returns code and message

CODE	EXPLANATION
1	Unrecognized command
2	Command only valid from program
3	Command not valid in program
4	Operand error
5	Input buffer full
6	Number out of range
7	Command not valid while running
8	Command not valid when not running
9	Variable error
10	Empty program line or undefined label
11	Invalid label or line number
12	Subroutine more than 16 deep
13	JG only valid when running in jog mode
14	EEPROM check sum error
15	EEPROM write error
16	IP incorrect sign during position move or IP given during forced deceleration
17	ED, BN and DL not valid while program running
18	Command not valid when contouring
19	Application strand already executing
20	Begin not valid with motor off
21	Begin not valid while running
22	Begin not possible due to Limit Switch
24	Begin not valid because no sequence defined
25	Variable not given in IN command

CODE	EXPLANATION
28	S operand not valid
29	Not valid during coordinated move
30	Sequence segment too short
31	Total move distance in a sequence > 2 billion
32	More than 511 segments in a sequence
33	VP or CR commands cannot be mixed with LI commands
41	Contouring record range error
42	Contour data being sent too slowly
46	Gear axis both master and follower
50	Not enough fields
51	Question mark not valid
52	Missing " or string too long
53	Error in {}
54	Question mark part of string
55	Missing [or []
56	Array index invalid or out of range
57	Bad function or array
58	Not a valid Command Operand (i.e._GNX)
59	Mismatched parentheses
60	Download error - line too long or too many lines
61	Duplicate or bad label
62	Too many labels
63	IF statement without ENDIF
65	IN command must have a comma
66	Array space full
67	Too many arrays or variables
71	IN only valid in task #0
80	Record mode already running
81	No array or source specified
82	Undefined Array
83	Not a valid number
84	Too many elements
90	Only X Y Z W valid operand
97	Bad binary command format
98	Binary Commands not valid in application program
99	Bad binary command number

CODE	EXPLANATION
100	Not valid when running ECAM
101	Improper index into ET (must be 0-256) (0~1024 with special firmaware)
102	No master axis defined for ECAM
103	Master axis modulus greater than 1024∗EP value
104	Not valid when axis performing ECAM
105	EB1 command must be given first
120	Bad Ethernet transmit
121	Bad Ethernet packet received
122	Ethernet input buffer overrun
123	TCP lost sync
124	Ethernet handle already in use
125	No ARP response from IP address
126	Closed Ethernet Handle Use IH
127	Illegal Modbus Function Code
128	IP Address Not valid

USAGE:

While Moving	Yes	Default Value	---
In a Program	Yes	Default Format	3.0
Command Line	Yes		
Can be Interrogated	Yes		
Used as an Operand	Yes		

OPERAND USAGE:

_TC contains the value of the error code.

EXAMPLES:

:GF32	Bad command
?TC	Tell error code
001	Unrecognized command

TD (Tell Dual (Auxiliary) Encoder)

[Status]

DESCRIPTION:

This command returns the current position of the dual (auxiliary) encoder.

ARGUMENTS: TD XYZW or ABCD

USAGE:

While Moving	Yes	Default Value	0
In a Program	Yes	Default Format	Position Format
Command Line	Yes		
Can be Interrogated	No		
Used as an Operand	Yes		

OPERAND USAGE:

_TDn contains the dual encoder position where n is an axis letter.

RELATED COMMANDS:

DE (Dual (Auxiliary) Encoder)

EXAMPLES:

:PF 7	Position format of 7
:TD	Return Dual encoder
0000200	
DUAL=_TDX	Assign the variable, DUAL, the value of TD

TE (Tell Error)

[Status]

DESCRIPTION:

This command returns the current position error of the motor. It is updated every servo cycle.

ARGUMENTS: TE XYZW or ABCD

USAGE:

While Moving	Yes	Minimum Value	-2147483648
In a Program	Yes	Maximum Value	2147483647
Command Line	Yes	Default Value	0
Can be Interrogated	No	Default Format	Position Format
Used as an Operand	Yes		

OPERAND USAGE:

_TEn contains the value of the position error where n is an axis letter.

RELATED COMMANDS:

ER (Error Limit)

#POSERR Excessive Position Error Special Label

EXAMPLES:

TE	Return position error
00005	
Error=_TEX	Sets the variable, Error, with the position error

HINT: Under normal operating conditions with servo control, the position error should be small. The position error is typically largest during acceleration and deceleration.

TH (Tell Handle)

[Status]

DESCRIPTION:

This command returns a formatted text display including the controllers MAC address, IP Address, and the IP address of the device connected to each of the handles. Also included are the port type and master / slave configuration.

This command is most useful from an external device, such as a terminal window or other program that can interpret the information.

ARGUMENTS: ***None***

USAGE:

While Moving	Yes	Default Value	n/a
In a Program	Yes	Default Format	formatted text
Command Line	Yes		
Can be Interrogated	No		
Used as an Operand	No		

RELATED COMMANDS:

IH (Internet Handle)

EXAMPLES:

TH	Tell Handle

TI (Tell Inputs)

[I/O]

DESCRIPTION:

This command returns the state of all 8 of the general digital inputs. Response is a decimal number which when converted to binary represents the status of all 8 digital inputs.

BIT	TI	PIN
Bit 7	Input 8	20
Bit 6	Input 7	7
Bit 5	Input 6	32
Bit 4	Input 5	19
Bit 3	Input 4	6
Bit 2	Input 3	31
Bit 1	Input 2	18
Bit 0	Input 1	5

ARGUMENTS: TI None

USAGE:

While Moving	Yes	Default Value	---
In a Program	Yes	Default Format	3.0
Command Line	Yes		
Can be Interrogated	Yes		
Used as an Operand	Yes		

OPERAND USAGE:

_TI contains the status byte of the input block. This can be masked to return only specified bit information.

EXAMPLES:

TI	
08	Input 4 is high, others low (0000 1000)
TI	
00	All inputs low (0000 0000)
Input =_TI	Sets the variable, Input, with the TI value
TI	
255	All inputs high (1111 1111)
SAC=”TI”	Send TI command to controller on handle C
VAR=_SAC	Store the returned value to a variable

TIME (Time Keyword)

[General]

DESCRIPTION:

The TIME operand contains the value of the internal free running, real time clock. The returned value represents the number of servo loop updates and is based on the TM (Time Base) command. The default value for the TM command is 1000. With this update rate, the operand TIME will increase by 1 count every update of approximately 1000usec. Note that a value of 1000 for the update rate (TM command) will actually set an update rate of 1/1024 seconds. Thus the value returned by the TIME operand will be off by 2.4% of the actual time.

The clock is reset to 0 with a standard reset or a master reset.

The keyword, TIME, does not require an underscore (_) as with the other operands.

USAGE:

Used as an Operand	Yes (without underscore)	Minimum value	0
Can be Interrogated	No	Maximum value	2147483647
		Format	TIME

EXAMPLES:

MG TIME	Display the value of the internal clock
Myvar = TIME	Assign TIME to Myvar
# Loop	Loop label
X = X + 1	Increment counter
JP # Loop, X, <500	Check if counter is less than 500
MG “Duration =”, TIME - Myvar	Print message

TK (Peak Torque Limit)

[Setting]

DESCRIPTION:

The TK commadn sets the peak torque limit on the motor commadn output and TL (Torque Limit) sets the continuous toque limit. When the average torque is below TI, the motor command signal can go up to the TK (peak Torque) for a short amount of time. If the TK is set lower than TL, then TL is tne maximum command output under all circumstances.

ARGUMENTS:

n is an unsigned number in the range of 0 to 9.99 volts.

n=0 disables the peak torque limit.

USAGE:

While Moving	Yes	Default Value	0
In a Program	Yes	Default Format	1.0
Command Line	Yes		

RELATED COMMANDS:

TL (Torque Limit)

OPERAND USAGE:

_TKn contains the value of the peak torque limit for the specified axis..

EXAMPLES:

TLA=7	Limit axis A to a 7-volt average torque output
TKA=9.99	Limit axis A to a 9.99-volt average torque output

TL (Torque Limit)

[Setting]

DESCRIPTION:

The TL command sets the limit on the motor command output. For example, TL of 5 limits the motor command output to 5 volts. Maximum output of the motor command is 9.998 volts.

Positive and negative torque cannot be set individually.

ARGUMENTS: TL x, y, z, w or TLX=x or TL a, b, c, d where

x, y z, w, or a, b, c, d are unsigned integers

USAGE:

While Moving	Yes	Minimum n Value	0
In a Program	Yes	Maximum n Value	9.9988
Command Line	Yes	Default Value	9.9988
Can be Interrogated	Yes	Default Format	1.4
Used as an Operand	Yes		

RELATED COMMANDS:

TK (Peak Torque Limit)

OPERAND USAGE:

_TLn contains the value of the torque limit where n is an axis letter.

EXAMPLES:

TL 1	Limit X-axis torque to 1volt
TL ?	Return torque limit
1.0000	

TM (Time Base)

[Configuration]

DESCRIPTION:

The TM command sets the sampling period of the control loop. Changing the sampling period will uncalibrate the speed and acceleration parameters. A negative number turns off the internal clock allowing for an external source to be used as the time base. The units of this command are µsec.

ARGUMENTS: TM n where

n is an integer in microseconds with a resolution of 125 microseconds.

USAGE:

While Moving	Yes	Minimum n Value	250
In a Program	Yes	Maximum n Value	20,000
Command Line	Yes	Default Value	1000
Can be Interrogated	Yes	Default Format	5.0
Used as an Operand	Yes		

OPERAND USAGE:

_TM contains the value of the sample time.

EXAMPLES:

TM 250	Set sample rate to 250 µsec (This will multiply all speeds by four and all acceleration by eight)
TM 1000	Return to default sample rate

***Note Although this manual refers to times in msec, think in terms of servo cycles. This includes everything from a WT (Wait) command to SP (Speed) commands.**

TN (Tangent)

[Configuration]

DESCRIPTION:

The TN m,n command describes the tangent axis to the coordinated motion path. m is the scale factor in counts/degree of the tangent axis. n is the absolute position of the tangent axis where the tangent axis is aligned with zero degrees in the coordinated motion plane. The tangent axis is specified with the VM (Coordinated Motion Mode) n,m,p command where p is the tangent axis. The tangent function is useful for cutting applications where a cutting tool must remain tangent to the part.

ARGUMENTS: TN m,n where

m is the scale factor in counts/degree, in the range between -127 and 127 with a fractional resolution of 0.004

When operating with stepper motors, m is the scale factor in steps / degree

n is the absolute position at which the tangent angle is zero, in the range between +/-2 * 10^9

USAGE:

While Moving	Yes	Default Value	---
In a Program	Yes	Default Format	---
Command Line	Yes		

OPERAND USAGE:

_TM contains the first position value for the tangent axis. This allows the user to correctly position the tangent axis before the motion begins.

RELATED COMMANDS:

VM (Coordinated Motion Mode)

CR (Circle)

EXAMPLES:

VM A,B,C	Specify coordinated mode for A and B-axis; C-axis is tangent to the motion path
TN 100,50	Specify scale factor as 100 counts/degree and 50 counts at which tangent angle is zero
VP 1000,2000	Specify vector position A,B
VE	End Vector
BGS	Begin coordinated motion with tangent axis

TP (Tell Position)

[Status]

DESCRIPTION:

This command returns the current position of the motor in quadrature counts.This value is updated every servo cycle.

ARGUMENTS: TP XYZW ABCD N

USAGE:

While Moving	Yes	Default Value	n/a
In a Program	Yes	Default Format	Position Format
Command Line	Yes		
Can be Interrogated	Yes		
Used as an Operand	Yes		
Virtual Axis	Yes		

OPERAND USAGE:

_TPn contains the current position value where n is an axis letter.

EXAMPLES:

:PF 7	Position format of 7
:TP	Return position
0000200	
PF-6.0	Change to hex format
TP	Return in hex
$0000C8	
Position=_TPX	Assign the variable, Position, the value of TP

TR (Trace Mode)

[Debug]

DESCRIPTION:

The TR command causes each instruction in a program to be sent out the communications port prior to execution. TR1 enables this function and TR0 disables it. The trace command is useful in debugging programs. It is not recommended to leave the TR command on for long durations (over 30 seconds) because it takes much longer to output the data from the controller than to execute it, hence, program execution will be affected. If no program lines are coming from the controller, issue “MG_XQn” or “MG_HXn” to see what line the controller is on. If the controller is at a trippoint, no lines will be output. Another way to take advantage of this command is to insert it in your program at a location previous to a suspected trouble spot (TR1) and just after the trouble spot (TR0). This way the trace will only show program lines that pertain to the debugging process.

ARGUMENTS: TR n where

n=0 or 1

0 disables function

1 enables function

No argument disables the trace function.

USAGE:

While Moving	Yes	Default Value	0
In a Program	Yes	Default Format	---
Command Line	Yes		
Can be Interrogated	No		
Used as an Operand	Yes		

TS (Tell Switches)

[Status]

DESCRIPTION:

TS returns the state of the Home switch, Forward and Reverse Limit switch, error conditions, motion condition and motor state. The value returned by this command is decimal and represents an 8 bit value (decimal value ranges from 0 to 255). Each bit represents the following status information.

Bit	Status
Bit 7	Axis in motion if high
Bit 6	Error limit exceeded if high
Bit 5	Motor off if high
Bit 4	Undefined
Bit 3	Forward Limit inactive if high
Bit 2	Reverse Limit inactive if high
Bit 1	State of home switch
Bit 0	Latch not armed if high

***Note The value for bits 1, 2 and 3 depend on the limit switch and home switch configuration (see the CN (Configure Limit Switches) command). For active low configuration (default), these bits are '1' when the switch is inactive and '0' when active. For active high configuration, these bits are '0' when the switch is inactive and '1' when active.**

ARGUMENTS: _TSx XYZW or ABCD

USAGE:

While Moving	Yes	Default Value	---
In a Program	Yes	Default Format	3.0
Command Line	Yes		
Can be Interrogated	Yes		
Used as an Operand	Yes		

OPERAND USAGE:

_TSX contains the current status of the switches.

EXAMPLES:

Assigns value of TS to the variable V1
V1=
015 (returned value)
Decimal value corresponding to bit pattern 00001111
X axis not in motion (bit 7 has value of 0)
X axis error limit not exceeded (bit 6 has value of 0)
X axis motor is on (bit 5 has value of 0)
X axis forward limit is inactive (bit 3 has value of 1)
X axis reverse limit is inactive (bit 2 has value of 1)
X axis home switch is high (bit 1 has value of 1)
X axis latch is not armed (bit 0 has value of 1)

TT (Tell Torque)

[Status]

DESCRIPTION:

The TT command reports the value of the analog servo command output signal, which is a number between -9.998 and 9.998 volts. This value is updated every servo cycle.

ARGUMENTS: TT XYZW or ABCD

USAGE:

While Moving	Yes	Minimum Value	-9.9988
In a Program	Yes	Maximum Value	9.9988
Command Line	Yes	Default Value	n/a
Can be Interrogated	Yes	Default Format	1.4
Used as an Operand	Yes		

OPERAND USAGE:

_TTn contains the value of the torque where n is an axis letter.

RELATED COMMANDS:

TL (Torque Limit)

TK (Peak Torque Limit)

EXAMPLES:

V1=_TT	Assigns value of TT to variable, V1
TT	Report torque
-0.2843	Torque is -.2843 volts

TV (Tell Velocity)

[Status]

DESCRIPTION:

The TV command returns the actual velocity in units of quadrature count/s. The value returned includes the sign. This value is averaged over 256 servo cycles.

ARGUMENTS: TV XYZW or ABCD

USAGE:

While Moving	Yes	Minimum Value	-12,000,000
In a Program	Yes	Minimum Value	12,000,000
Command Line	Yes	Default Value	n/a
Can be Interrogated	Yes	Default Format	8.0
Used as an Operand	Yes		
Virtual Axis	No		

OPERAND USAGE:

_TVn contains the value for the velocity where n is an axis letter.

EXAMPLES:

VELX=_TVX	Assigns value of velocity to the variable VELX
TVX	Returns the velocity of the X axis
0003420	

TW (Time Wait)

[Setting]

DESCRIPTION:

The TW n command sets the timeout in msec to declare an error if the MC (Motion Complete) command is active and the motor is not at or beyond the actual position within n msec after the completion of the motion profile. If a timeout occurs, then the MC trippoint will clear and the stopcode will be set to 99. An application program will jump to the special label #MCTIME. If included, the RE (Return from Error) command should be used to return from the #MCTIME subroutine.

ARGUMENTS: TW x, y, z, w or TWX=x or TW a, b, c, d where

x, y z, w, or a, b, c, d are signed integers

n specifies timeout in msec, -1 disables the timeout

USAGE:

While Moving	Yes	Minimum n Value	-1
In a Program	Yes	Maximum n Value	32766
Command Line	Yes	Default Value	32766
Can be Interrogated	Yes	Default Format	
Used as an Operand	Yes		
Virtual Axis	No		

OPERAND USAGE:

_TW contains the timeout in msec for the MC command where n is an axis letter.

RELATED COMMANDS:

MC (Motion Complete)

TY (Tell Yaskawa Absolute Encoder)

[Status]

DESCRIPTION:

TY (Tell Yaskawa Absolute Encoder) reports the position that was read when the absolute encoder data was requested using the AE (Absolute Encoder) command..

ARGUMENTS: TY XYZW TP ABCD

USAGE:

While Moving	Yes	Default Value	-2,147,483,648
In a Program	Yes	Default Format	10.0
Not in a Program	Yes		
Can be Interrogated	Yes		
Used as an Operand	Yes		

RELATED COMMANDS:

AE (Absolute Encoder)

QY (Query Yaskawa Absolute Encoder Alarm)

EXAMPLES:

MOX	Motor Off
AEX=4096	Read X absolute encoder
DPX=_TPX+XhmOfs	Add offset variable to current absolute position
TPX	Tell current defined position
TYX	Tell absolute encoder position (when it was read)
SHX	Enable servo (Servo Here)

UL (Upload)

[General]

DESCRIPTION:

The UL command transfers data from the SMC–4000 to a host computer. Programs are sent without line numbers. The Uploaded program will be followed by a <control>Z or a \ as an end of Text marker.

ARGUMENTS: None

USAGE:

While Moving	Yes	Default Value	n/a
In a Program	No	Default Format	n/a
Command Line	Yes		
Can be Interrogated	No		
Used as an Operand	Yes		

OPERAND USAGE:

When used as an operand, _UL gives the number of available variables. The total number of variables is 510.

RELATED COMMAND:

DL (Download)

EXAMPLES:

UL;	Begin upload
#A	Line 0
NO This is an Example	Line 1
NO Program	Line 2
EN	Line 3
<cntrl>Z	Terminator

VA (Vector Acceleration)

[Motion]

DESCRIPTION:

This command sets the acceleration rate of the vector in a coordinated motion sequence.

ARGUMENTS: VA s,t where

s and t are unsigned integers in the range 1024 to 68,431,360. s represents the vector acceleration for the S coordinate system and t represents the vector acceleration for the T coordinate system. The parameter input will be rounded down to the nearest factor of 1024. The units of the parameter is counts per second squared.

USAGE:

While Moving	Yes	Minimum n Value	1024
In a Program	Yes	Maximum n Value	67107840
Command Line	Yes	Default Value	256000
		Default Format	Position Format

OPERAND USAGE:

_VAx contains the value of the vector acceleration where x is S or T.

RELATED COMMANDS:

VS (Vector Speed)

VP (Vector Position)

VE (Vector Sequence End)

CR (Circle)

VM (Coordinated Motion Mode)

BG (Begin)

VD (Vector Deceleration)

VT (Vector Time Constant)

EXAMPLES:

VA 1024	Set vector acceleration to 1024 counts/sec^2
MG_VA	Return vector acceleration
00001024	
VA 20000	Set vector acceleration
MG_VA	
0019456	Return vector acceleration
ACCEL=_VA	Assign variable, ACCEL, the value of VA

VD (Vector Deceleration)

[Motion]

DESCRIPTION:

This command sets the deceleration rate of the vector in a coordinated motion sequence.

ARGUMENTS: VD s,t where

s and t are unsigned integers in the range 1024 to 68,431,360. s represents the vector deceleration for the S coordinate system and t represents the vector deceleration for the T coordinate system. The parameter input will be rounded down to the nearest factor of 1024. The units of the parameter is counts per second squared.

USAGE:

While Moving	No	Minimum n Value	1024
In a Program	Yes	Maximum n Value	67107840
Command Line	Yes	Default Value	256000
Controller Usage	Yes	Default Format	Position Format

OPERAND USAGE:

_VDx contains the value of the vector deceleration where x is S or T.

RELATED COMMANDS:

VA (Vector Acceleration)

VS (Vector Speed)

VP (Vector Position)

CR (Circle)

VE (Vector Sequence End)

VM (Coordinated Motion Mode)

BG (Begin)

VT (Vector Time Constant)

EXAMPLES:

#VECTOR	Vector Program Label
VMXY	Specify plane of motion
VA1000000	Vector Acceleration
VD 5000000	Vector Deceleration
VS 2000	Vector Speed
VP 10000, 20000	Vector Position
VE	End Vector
BGS	Begin Sequence

VE (Vector Sequence End)

[Motion]

DESCRIPTION:

VE is required to specify the end segment of a coordinated move sequence. VE follows the final VP (Vector Position) or CR (Circle) command in a sequence. VE is equivalent to the LE (Linear Interpolation End) command. The VE command will apply to the selected coordinate system, S or T. To select the coordinate system, use the command CAS or CAT. If a VE command is not issued before the controller runs all the linear segments, motion will stop instantaneously.

ARGUMENTS: VEx ***where x is S or T***

No argument specifies the end of a vector sequence.

USAGE:

While Moving	Yes	Default Value	-
In a Program	Yes	Default Format	-
Command Line	Yes		

OPERAND USAGE:

_VEx contains the length of the vector in counts where x is S or T.

RELATED COMMANDS:

VM (Coordinated Motion Mode)

VS (Vector Speed)

VA (Vector Acceleration)

VD (Vector Deceleration)

VP (Vector Position)

BG (Begin)

CS (Clear Sequence)

EXAMPLES:

VM XY	Vector move in XY
VP 1000,2000	Linear segment
VP 0,0	Linear segment
VE	End sequence
BGS	Begin motion

VF (Variable Format)

[General]

DESCRIPTION:

The VF command allows the variables and arrays to be formatted for number of digits before and after the decimal point. When displayed, the value m represents the number of digits before the decimal point, and the value n represents the number of digits after the decimal point. When in hexadecimal, the string will be preceded by a $. Hex numbers are displayed as 2's complement with the first bit used to signify the sign.

If a number exceeds the format, the number will be displayed as the maximum possible positive or negative number (i.e. 999.99, -999, $8000 or $7FF).

ARGUMENTS: VF m.n where

m and n are unsigned numbers. A negative m specifies hexadecimal format.

USAGE:

While Moving	Yes	Minimum m Value	-8
In a Program	Yes	Maximum m Value	10
Command Line	Yes	Default m Value	10
Can be Interrogated	Yes	Minimum n Value	0
Used as an Operand	Yes	Maximum n Value	4
		Default n Value	4
		Default Format	2.1

OPERAND USAGE:

_VF contains the value of the format for variables and arrays.

EXAMPLES:

VF 5.3	Sets 5 digits of integers and 3 digits after the decimal point
VF 8.0	Sets 8 digits of integers and no fractions
VF -4.0	Specify hexadecimal format with 4 bytes to the left of the decimal

VM (Coordinated Motion Mode)

[Motion]

DESCRIPTION:

The VM command specifies the coordinated motion mode and the plane of motion. This mode may be specified for motion on any set of two axes.

The motion is specified by the instructions VP (Vector Position) and CR (Circle), which specify linear and circular segments. Up to 511 segments may be given before the Begin Sequence (BGS or BGT) command. Additional segments may be given during the motion when the buffer frees additional spaces for new segments. It is the responsibility of the user to keep enough motion segments in the buffer to ensure continuous motion.

The VE (Vector Sequence End) command must be given after the last segment. This allows the controller to properly decelerate.

The VM command will apply to the selected coordinate system, S or T. To select the coordinate system, use the command CAS or CAT.

ARGUMENTS: VM n,m,p where

n and m specify plane of vector motion and can be any two axes. Vector Motion can be specified for one axis by specifying second parameter, m, as N. Specifying one axis is useful for obtaining sinusoidal motion on one axis.

p is the tangent axis and can be specified as any axis. A value of N for the parameter, p, turns off tangent function.

USAGE:

While Moving	No	Minimum n Value	0.0001
In a Program	Yes	Maximum n Value	10
Command Line	Yes	Default Value	A,B
		Default Format	-

OPERAND USAGE:

_VMn contains instantaneous commanded vector velocity for the specified coordinate system, S or T.

RELATED COMMANDS:

VP (Vector Position)

VS (Vector Speed)

VA (Vector Acceleration)

VD (Vector Deceleration)

CR (Circle)

VE (Vector Sequence End)

CS (Clear Sequence)

VT (Vector Time Constant)

AV (After Vector Distance)

EXAMPLES:

CAS	Specify S coordinate system
VM A,B	Specify coordinated mode for A,B
CR 500,0,180	Specify arc segment
VP 100,200	Specify linear segment
VE	End vector
BGS	Begin sequence

VP (Vector Position)

[Motion]

DESCRIPTION:

The VP command defines the target coordinates of a straight line segment in a 2 axis motion sequence which have been selected by the VM command. The units are in quadrature counts, and are a function of the vector scale factor set using the VS (Vector Speed) command.

For three or more axes linear interpolation, use the LI (Linear Interpolation Distance) command.

The VP command will apply to the selected coordinate system, S or T. To select the coordinate system, use the command CAS or CAT.

ARGUMENTS: VP n,m <o> p where

n and m are signed integers in the range -2147483648 to 2147483647. The length of each segment must be limited to 8 x 10^6. The values for n and m will specify a coordinate system from the beginning of the sequence.

o specifies a vector speed to be taken into effect at the execution of the vector segment. n is an unsigned even integer between 0 and 12,000,000 for servo motor operation and between 0 and 3,000,000 for stepper motors.

p specifies a vector speed to be achieved at the end of the vector segment. p is an unsigned even integer between 0 and 8,000,000.

USAGE:

While Moving	Yes	Default Value	-
In a Program	Yes	Default Format	-
Command Line	Yes		
Virtual Axis	Yes		

OPERAND USAGE:

_VPn contains the absolute coordinate of the axes at the last intersection along the sequence. For example, during the first motion segment, this instruction returns the coordinate at the start of the sequence. The use as an operand is valid in the linear mode, LM, and in the Vector mode, VM.

RELATED COMMANDS:

CR (Circle)

VM (Coordinated Motion Mode)

VA (Vector Acceleration)

VD (Vector Deceleration)

VE (Vector Sequence End)

VS (Vector Speed)

BG (Begin)

VT (Vector Time Constant)

EXAMPLES:

#A	Program A
VM	Specify motion plane
VP 1000,2000	Specify vector position A,B
CR 1000,0,360	Specify arc
VE	Vector End
VS 2000	Specify vector speed
VA 400000	Specify vector acceleration
BGS	Begin motion sequence
EN	End program

***Note The first vector in a coordinated motion sequence defines the origin for that sequence. All other vectors in the sequence are defined by their endpoints with respect to the start of the move sequence.**

VR (Vector Speed Ratio)

[Motion]

DESCRIPTION:

The VR sets a ratio to be used as a multiplier of the current vector speed. The vector speed can be set by the command VS (Vector Speed) or the operators < and > used with CR (Circle), VP (Vector Position) and LI (Linear Interpolation Distance) commands. VR takes effect immediately and will ratio all the following vector speed commands. VR doesn't ratio acceleration or deceleration, but the change in speed is accomplished by accelerating or decelerating at the rate specified by VA (Vector Acceleration) and VD (Vector Deceleration).

ARGUMENTS: VR s,t where

s and t are between 0 and 10 with a resolution of .0001. The value specified by s is the vector ratio to apply to the S coordinate system and t is the value to apply to the T coordinate system.

USAGE:

While Moving	Yes	Minimum n Value	0.0001
In a Program	Yes	Maximum n Value	10
Command Line	Yes	Default Value	1
		Default Format	-

OPERAND USAGE:

_VRn contains the vector speed ratio of the specified coordinate system, S or T.

RELATED COMMANDS:

VS (Vector Speed)

EXAMPLES:

#A	Vector Program
VMXY	Vector Mode
VP 1000,2000	Vector Position
VE	End Sequence
VS 2000	Vector Speed
BGS	Begin Sequence
AMS	After Motion
JP#A	Repeat Move
#SPEED	Speed Override
VR ratio	Set ratio according to variable
JP#SPEED	Loop
XQ#A,0; XQ#SPEED,1	Execute task 0 and 1 simultaneously

***Note UseVR for feedrate override, when specifying the speed of individual segments using the operator '<'.**

VS (Vector Speed)

[Motion]

DESCRIPTION:

The VS command specifies the speed of the vector in a coordinated motion sequence in either the LM (Linear Interpolation Mode) or VM (Coordinated Motion Mode) modes. VS may be changed during motion.

Vector Speed can be calculated by taking the square root of the sum of the squared values of speed for each axis specified for vector or linear interpolated motion.

ARGUMENTS: VS s,t where

s and t are unsigned even numbers in the range 2 to 8,000,000 for servo motors and 2 to 3,000,000 for stepper motors. s is the speed to apply to the S coordinate system and t is the speed to apply to the T coordinate system. The units are counts per second.

USAGE:

While Moving	Yes	Default Value	25000
In a Program	Yes	Default Format	---
Command Line	Yes		

OPERAND USAGE:

_VSn contains the vector speed of the specified coordinate system, S or T.

RELATED COMMANDS:

VA (Vector Acceleration)

VP (Vector Position)

CR (Circle)

LI (Linear Interpolation Distance)

VM (Coordinated Motion Mode)

BG (Begin)

VE (Vector Sequence End)

EXAMPLES:

VS 2000	Define vector speed of S coordinate system
MG_VSS	Return vector speed of S coordinate system
002000	

Hint: ***Vector speed can be attached to individual vector segments. For more information, see description of VP (Vector Position), CR (Circle), and LI (Linear Interpolation Distance) commands.***

VT (Vector Time Constant)

[Motion]

DESCRIPTION:

The VT command filters the acceleration and deceleration functions in vector moves of VM (Coordinated Motion Mode), LM (Linear Interpolation Mode) type to produce a smooth velocity profile. The resulting profile, known as Smoothing, has continuous acceleration and results in reduced mechanical vibrations. VT sets the bandwidth of the filter, where 1 means no filtering and 0.004 means maximum filtering. Note that the filtering results in longer motion time.

ARGUMENTS: VT s,t where

s and t are unsigned numbers in the range between 0.004 and 1.0, with a resolution of 1/256. The value s applies to the S coordinate system and t applies to the T coordinate system.

USAGE:

While Moving	Yes	Minimum n Value	0.004
In a Program	Yes	Maximum n Value	1.000
Command Line	Yes	Default Value	1.0
Controller Usage	All controllers	Default Format	1.4

OPERAND USAGE:

_VTn contains the vector time constant, for the specified coordinate plane, where n is S or T.

RELATED COMMANDS:

IT (Independent Time Constant)

EXAMPLES:

VT 0.8	Set vector time constant for S coordinate system
MG _VTT	Return vector time constant for T coordinate system
0.8	

WC (Wait for Contour)

[Program Flow]

DESCRIPTION:

The WC command acts as a flag in the Contour Mode. After this command is executed, the controller does not receive any new data until the internal contour data buffer is ready to accept new commands. This command prevents the contour data from overwriting itself in the contour data buffer.

USAGE:

While Moving	Yes	Default Value	---
In a Program	Yes	Default Format	---
Command Line	Yes		
Can be Interrogated	No		
Used as an Operand	No		

RELATED COMMANDS:

CM (Contour Mode)

CD (Contour Data)

DT (Delta Time)

EXAMPLES:

CM	Specify contour mode
DT 4	Specify time increment for contour
CD 200	Specify incremental position
WC	Wait for contour data to complete
CD 100	
WC	Wait for contour data to complete
DT 0	Stop contour
CD 0	Exit mode

WH (Which Handle)

[Status]

DESCRIPTION:

The WH command is used to identify the handle in which the command is executed. The command returns IHA through IHH to indicate on which handle the command was executed. The command returns RS232 if communicating serially.

ARGUMENTS: ***None***

USAGE:

While Moving	Yes	Default Value	---
In a Program	Yes	Default Format	---
Command Line	Yes		

RELATED COMMANDS:

TH (Tell Handle)

OPERAND USAGE:

_HW contains the numeric representation of the handle in which a command is executed. Handles A through H are indicated by the value 0-7, while a -1 indicates the serial port.

EXAMPLES:

“WH”	Request handle identification
“IHC”	Command executed in handle C
“WH”	Request handle identification
RS232	Command executed in RS232 port

WT (Wait)

[Trippoint]

DESCRIPTION:

The WT command is a trippoint used to time events. After this command is executed, the controller will wait for the number of samples specified before executing the next command. If the TM (Time Base) command has not been used to change the sample rate from 1 msec, then the units of the Wait command are milliseconds.

ARGUMENTS: WT n where

n is an unsigned integer

USAGE:

While Moving	Yes	Minimum Value	0
In a Program	Yes	Maximum Value	2147483647
Command Line	Yes	Default Value	---
Can be Interrogated	No	Default Format	---
Used as an Operand	No		

EXAMPLES:

Assume that 10 seconds after a move is over a relay must be closed.

#A	Program A
PR 50000	Position relative move
BG	Begin the move
AM	After the move is over
WT 10000	Wait 10 seconds
SB 1	Turn on relay
EN	End Program

XQ (Execute Program)

[General]

DESCRIPTION:

The XQ command begins execution of a program residing in the program memory of the controller. Execution will start at the label or line number specified. If the cmmand is issued from an external source, all argumetns can be omitted, and the controller will execute the first line of code as thread 0.

ARGUMENTS: XQ #A,n XQm,n where

A is a program label of up to seven characters

m is a line number

n is the thread number 0 through 7

USAGE:

While Moving	Yes	Default Value	n = 0
In a Program	Yes	Default Format	---
Command Line	Yes		
Can be Interrogated	No		
Used as an Operand	Yes		

OPERAND USAGE:

_XQn contains the current line number of execution for thread n, and -1 if thread n is not running.

RELATED COMMANDS:

HX (Halt Execution)

EXAMPLES:

XQ #Apple,0	Start execution at label Apple, thread zero
XQ #data,3	Start execution at label data, thread three
XQ 0	Start execution at line 0

ZS (Zero Subroutine Stack)

[Program Flow]

DESCRIPTION:

The ZS command is only valid from within an application program and is used to avoid returning from an interrupt (either input or error). ZS alone returns the stack to its original condition. ZS1 adjusts the stack to eliminate one return. ZS acts on the stack of the program thread in which the command was executed. The ZS command is typically used in special label error routines. Normal program flow should not require the use of ZS.

ARGUMENTS: ZS n where

0 returns stack to original condition

1 eliminates one return on stack

USAGE:

While Moving	Yes	Minimum value	0
In a Program	Yes	Maximum value	15
Command Line	No	Default Value	n/a
Can be Interrogated	Yes	Default Format	n/a
Used as an Operand	Yes		

OPERAND USAGE:

_ZSn contains the stack level for the specified thread where n = 0-7. The response indicates zero for beginning condition and 15 for the deepest value.

EXAMPLES:

II1	Input Interrupt on 1
#A;JP #A;EN	Main program
#ININT	Input Interrupt
MG "INTERRUPT"	Print message
S=_ZS	Interrogate stack
S=	Print stack
ZS	Zero stack
S=_ZS	Interrogate stack
S=	Print stack
EN	End

SMC–4000 **Command Interrogation List**

Command	Definition	units	min	max	**default**
_AB	Status of abort input	status	0=Aborted	1=OK	**n/a**
_ACx	Axis acceleration rate	counts/sec2	1024	67107840	**256000**
_AEx	"Last absolute encoder read (0=X, 1=Y, etc)"	code	0	3	n/a
_ALx	High speed position capture status	status	0=TRIPPED	1=NOT YET	**0**
_AV	Distance from the start of vector sequence	counts	0	2147483647	**0**
_BGx	Is axis in motion?	status	0=NO	1=YES	**0=NO**
_BK	The program debug breakpoint	line number	-2000	2000	0
_BLx	Reverse software limit	counts	-2147483648	2147483647	**-2147483648**
_BN	Serial number of the SMC–4000	n/a	1	65535	**n/a**
_CEx	Type of encoder selected	configuration	0	15	**0**
_CF	Returns the default port that unsolicited messages are directed to (ASCII)	configuration	65 = 'A'	83 = 'S'	**83 = 'S'**
_CM	Is the contour mode buffer full?	status	0=NO	1=YES	**0=NO**
_CN	Returns the configuration of the limit switches	configuration	-1 = Active Low	1 = Active High	-1 = Active Low
_CN1	Returns the configuration of the home input	configuration	-1 = Active Low	1 = Active High	-1 = Active Low
_CN2	Returns the configuration of the latch input	configuration	-1 = Active Low	1 = Active High	-1 = Active Low
_CN3	Returns the configuration of the selective abort function	configuration			
_CS	Current segment number for Vector Mode	segment	0	511	**0**
_CW	Port #1 data adjustment (MG from prog, chars have bit 8 set)	status	1=SET	2=OFF	**2=OFF**
_DA	Number of available arrays	quantity	0	30	**30**
_DCx	Axis deceleration rate	counts/sec2	1024	67107840	**256000**
_DEx	Encoder position of the auxiliary encoder	counts	-2147483648	2147483647	**n/a**
_DL	Number of available labels	quantity	0	510	**510**
_DM	Number of available array locations	quantity	0	2000	**2000**
_DPx	Current encoder position of axis	counts	-2147483648	2147483647	**n/a**
_DT	Time interval for contour mode	2N mSec	0	8	**0**
_DVx	Is the axis using dual loop PID?	status	0=NO	1=YES	**0=NO**
_EB	Is CAM mode enabled?	status	0=NO	1=YES	**0=NO**
_EC	returns the current index into the cam table	pointer	0	1024	**0**
_ED	The last line that caused a CMDERR	line number	0	999	**n/a**
_EGx	Is CAMMING axis engaged?	status	0=NO	1=YES	**0=NO**
_EMx	Cam cycle for camming (master or slave)	counts	0	2147483647	**0**
_EO	Is echo mode on?	status	0=NO	1=YES	**0=NO**
_EP	CAMMING interval (resolution)	counts	1	32767	**256**
_EQx	Status of ECAM slave	status	0	3	**0**
_ERx	Axis following error limit	counts	0	32767	**16384**
_ES	Ellipse scale ratio	n/a	0.0001	1	**1**
_FAx	Axis acceleration feedforward	constant	0	8191	**0**
_FLx	Forward software limit	counts	-2147483648	2147483647	**2147483647**
_FVx	Axis velocity feedforward	constant	0	8191	**0**
_GRx	Gear ratio of the axis	constant	-127.9999	127.9999	**0**
_HMx	State of the home switch	status	0=ACTIVE	1=INACTIVE	**n/a**
_HXx	Thread info (x is thread 0 through 3)	0=NOT RUNNING	1=RUNNING	2=AT TRIPPOINT	**n/a**
_IA	Returns the IP address as a 32 bit signed number	address	0	2147483647	**0**
_IA1	Returns the Ethernet retry time	mSec	0	2147483647	**250**
_IA2	Returns the number of available handles	handles	0	8	**8**
_IA3	Returns the number of the handle using this operand	handle	0	5	**n/a**
_IHh0	Returns the IP address as a 32 bit signed number ("h" is handle "A" - "H")	address	-2147483648	2147483647	**-1**

SMC–4000 **Command Interrogation List**

Command	Definition	units	min	max	**default**
_IHh1	Returns the slave port number	number	0	65535	**0**
_IHh2	Returns the handle status (See IH command description)	status	-2	2	**0**
_IHh3	Returns ARP status	status	0 = Successful	1 = Failed	**n/a**
_II	Returns the bitmask of all inputs that are selected as interrupts	configuration	0	127	**0**
_ILx	Integrator limit of the axis	voltage	-9.9988	9.9988	**9.9988**
_IPX	Current encoder position of axis	counts	-2147483648	2147483647	**n/a**
_ITx	S curve smoothing function value	constant	0.004	1	**1**
_JGx	Jog speed for that axis	counts/sec	0	8000000	**25000**
_KDx	Derivative Constant for PID loop	constant	0	4095.875	**64**
_KIx	Integrator for PID loop	constant	0	2047.875	**0**
_KPx	Proportional Constant for PID loop	constant	0	1023.875	**6**
_LC	Status of Lock Controller command	configuration	0 = UNLOCKED	2 = ALL LOCKED	**0-Jan**
_LE	Length of the vector	counts	0	2147483647	**0**
_LFx	Forward Limit Switch	status	0 = ACTIVE	1 = INACTIVE	**n/a**
_LM	Number of free locations in linear mode buffer	n/a	0	511	**n/a**
_LRx	Reverse Limit Switch	status	0 = ACTIVE	1 = INACTIVE	**n/a**
_LS	The total number of program lines	lines	0	999	**0**
_LTx	Stop distance set in the Latch Target Command	counts	-2147483648	2147483647	**0**
_LZ	Serial port leading zero removal	status	0 = OFF	1 = ON	**0**
_MW	Returns the current configuration of the Modbus Wait Command	configuration	0 = OFF	1 = ON	**1**
_MOx	Current state of motor, enabled or not	status	0=ENABLED	1=DISABLED	**0 / 1**
_MTx	Type of motor	configuration	-1	1	**1**
_NBx	Returns the Notch Filter Bandwidth	Hertz	0	62	**0.5**
_NFx	Returns the Notch Frequency	Hertz	0	255	**0**
_NZx	Returns the Notch Zero	Hertz	0	62	**0**
_OEx	Indicates if servo enable signal will shut off if "_Erx" is exceeded	status	0=NO	1=YES	**0=NO**
_OFx	Axis command offset	voltage	-9.9988	9.9988	**0**
_OPx	Entire byte or word of output port (x = output bank 0-3)	byte or word	0	65535	**0**
P1CD	Status code of serial port	status	-1	3	**n/a**
P1CH	The last character received from serial port	character	0	255	**n/a**
P1NM	The last number received from serial port	number	-2147483648	2147483647	**n/a**
P1ST	The last string received from serial port	string		6 chars max	**n/a**
_PAx	Last commanded absolute position if moving, otherwise current position	counts	-2147483648	2147483647	**0**
_PF	Encoder position format (see PF command)	configuration	-8.4	10.4	**10.4**
_PRx	Current incremental distance to move (Even if move set by PA)	counts	-2147483648	2147483647	**0**
_QAx	The unmodularized position of the aux encoder	counts	-2147483648	2147483647	n/a
_QPx	The unmodularized position of the main encoder	counts	-2147483648	2147483647	
_RC	Status of record mode	status	0= NOT RECORDING	1=RECORDING	**0= NOT RECORDING**
_RD	Array index that record mode will use next	index	0	7999	**0**
_RLx	Encoder value of last latched position	counts	-2147483648	2147483647	**0**
_RPx	Current commanded position of the motor	counts	-2147483648	2147483647	**0**
_RUx	The unmodularized latch of the main encoder	counts	-2147483648	2147483647	n/a
_SCx	The Stop Code of the axis	code	0	150	**1**
_SPx	Speed parameter of the axis	counts/sec	0	8000000	**25000**
_TB	Status information from controller	byte	0	255	**1**
_TC1	Error code and message from controller	number	0	255	**0**
_TDx	Current auxiliary encoder position	counts	-2147483648	2147483647	**n/a**
_TEx	Difference between commanded & actual axis position	counts	-2147483648	2147483647	**n/a**

SMC–4000 **Command Interrogation List**

Command	Definition	units	min	max	**default**
_TIx	8 inputs as a decimal or hex value (x = input bank 0-7)	byte	0	255	**n/a**
TIME	Counter since SMC powered on or reset	servo cycles	0	2147483647	**0**
_TKx	The value of the peak torque limit	volts	0	9.9988	0
_TLx	Torque limit of axis	voltage	0	9.9988	**9.9988**
_TM	Servo update cycle for all axes	uSec	250	20000	**1000**
_TN	Position of first tangent point	counts	-2147483648	2147483647	**0**
_TPx	Current encoder position of axis	counts	-2147483648	2147483647	**n/a**
_TSx	Status of switches for axis	byte	0	255	**n/a**
_TTx	Current output voltage to amplifier	voltage	-9.9988	9.9988	**0**
_TVx	Velocity of axis (averaged over 256 servo cycles)	counts/sec	0	8000000	**n/a**
_TWx	Time limit that program will wait for axis to get to target position (MCx)	milliseconds	-1	32766	**32766**
_TYx	Report the value of the absolute encoder at power up	counts	-2147483648	2147483647	-2147483648
_UL	Number of variables available	n/a	0	510	**510**
_VA	acceleration value for vector mode	counts/sec2	1024	68431360	**256000**
_VD	Deceleration value for vector mode	counts/sec2	1024	68431360	**256000**
_VE	Length of vector (all moves in coordinated move sequence)	counts	0	2147483647	**0**
_VF	Setting of variable formatting	n/a	0	10.4	**10.4**
_VM	Number of free locations in vector mode buffer	n/a	0	511	**511**
_VPx	Absolute coordinate of the axis in the last segment	counts	-2147483648	2147483647	**0**
_VR	Vector speed ratio	n/a	0	10	**1**
_VS	Vector Speed	counts/sec	2	8000000	**25000**
_VT	S curve smoothing value for vector mode	constant	0.004	1	**1**
_XQx	Current line number being executed (x = thread #)	line number	-1	1999	**n/a**
_ZS	Current subroutine depth	number	0	16	**n/a**

5 Programming Basics

Introduction

The SMC–4000 provides over 100 commands for specifying motion and machine parameters. Commands are included to initiate action, interrogate status and configure the digital filter.

The SMC–4000 instruction set is BASIC-like and easy to use. Instructions usually consist of two uppercase letters that normally correspond phonetically with the appropriate function. For example, the instruction BG begins motion, and ST stops motion.

Commands can be sent "live" for immediate execution by the SMC–4000, or an entire group of commands (a program) can be downloaded into the SMC–4000 memory for execution at a later time. Combining commands into groups for later execution is referred to as Applications Programming and is discussed in the following chapter.

This section describes the SMC–4000 instruction set and syntax. A complete listing of all SMC–4000 instructions is included in the command reference section.

Program Maximums

Commands per line	Until 80 characters
Labels among all threads	510
Lines among all threads	2000
Subroutine nesting level	16
Threads	8

Command Syntax

SMC–4000 instructions are represented by two ASCII upper case characters followed by applicable arguments. A space may be inserted between the instruction and arguments. A semicolon or <enter> is used to terminate the instruction for processing by the SMC–4000 command interpreter.

IMPORTANT: All SMC–4000 commands must be upper case.

For example, the command

PR 4000 <enter> Position Relative

PR is the two character instruction for Position Relative. 4000 is the argument which represents the required position value in counts. The <enter> terminates the instruction. The spaces between commands and argumants are always optional.

For specifying data for the X,Y,Z and W axes, commas are used to separate the axes and preserve axis order as X,Y,Z and W. If no data is specified for an axis, a comma is still needed as shown in the examples below. If no data is specified for an axis, the previous value is maintained. The space between the data and instruction is optional. For the SMC–4000, the four axes are referred to as A,B,C,D where X,Y,Z,W and A,B,C,D may be used interchangeably.

To view the current values for each command, specify the command followed by a ? for each axis requested. The SMC–4000 provides an alternative method for specifying data.

Here data is specified individually using a single axis specified such as X,Y,Z or W (or A,B,C,or D for the SMC–4000). An equal sign is used to assign data to that axis. For example:

PRZ=1000 Sets the Z axis data as 1000

All axes data may be specified at once using the * symbol. This sets all axes to have the same data. For example:

PR*=1000 Sets all axes to 1000

Example XYZW Syntax for Specifying Data

PR*=1000	Specify data on all axes as 1000
PRY=1000	Specify Y as 1000
PR 1000	Specify X only as 1000
PR ,2000	Specify Y only as 2000
PR ,,3000	Specify Z only as 3000
PR ,,,4000	Specify W only as 4000
PR 2000,4000,6000,8000	Specify X,Y,Z, and W
PR ,8000,,9000	Specify Y and W only
PR ,?	Request Y value only

Instead of data, some commands request action to occur on an axis or group of axes. For example, ST XY stops motion on both the X and Y axes. Commas are not required in this case since the particular axis is specified by the appropriate letter X Y Z or W. If no parameters follow the instruction, action will take place on all axes. The letter S or T is used to specify a coordinated motion sequence.

Example XYZW syntax for Requesting Action

BG X	Begin X only
BG Y	Begin Y only
BG XYZW	Begin all axes
BG YW	Begin Y and W only
BG	Begin all axes
BG S	Begin coordinated sequence
BG SW	Begin coordinated sequence and W axis
BG ABCD	Begin all axes
BG D	Begin D only

Controller Response to Commands

For each valid command entered, the SMC–4000 returns a colon (:). If the SMC–4000 decodes a command as invalid, it returns a question mark (?).

***Note The SMC–4000 returns a : for valid commands.**

***Note The SMC–4000 returns a ? for invalid commands.**

For example, if the command bg is sent in lower case, the SMC–4000 will return a ?.

:bg <enter>	Invalid command (lower case)
?	SMC–4000 returns a ?

VERY IMPORTANT!

The command Tell Code, TC1, will return the reason for the "?" received for the last invalid command.

:TC1 <enter>	Tell Code command
1 Unrecognized command	Returned response

There are several coded reasons for receiving a ?. Example codes include unrecognized command (such as typographical entry or lower case), a command given at improper time, or a command out of range, such as exceeding maximum speed. A complete listing of all codes is listed in the TC command in the Command Reference section.

For interrogation instructions such as TP (Tell Position) or TS (Tell Switches) , the SMC–4000 returns the requested data on the next line followed by a carriage return and line feed. The data returned is in decimal format.

Tell Position X	:TP X <enter>
data returned	0000000000
Tell Position X and Y	:TP XY <enter>
data returned	0000000000,0000000000

The format of the returned data can be set using the PF (Position Format) and VF (Variable Format) command.

:PF 4 <enter>	Position Format is 4 integers
:TP X <enter>	Tell Position
0000	returned data

Command Summary

Each SMC–4000 command is described fully in the command reference section of this manual. A summary of the commands follows.

The commands are grouped in this summary by the following functional categories:

- Motion
- Program Flow
- General Configuration
- Control Settings
- Status and Error/Limits

Motion commands are those to specify modes of motion such as Jog Mode or Linear Interpolation, and to specify motion parameters such as speed, acceleration and deceleration, and distance.

Program flow commands are used in Application Programming to control the program sequencer. They include the jump on condition command and event triggers such as after position and after elapsed time.

General configuration commands are used to set controller configurations such as setting and clearing outputs, formatting variables, and motor/encoder type.

The control setting commands include filter settings such as KP (Proportional Constant), KD (Derivative Constant), and KI (Integrator) and sample time.

Error/Limit commands are used to configure software limits and position error limits.

Motion

AB	Abort Motion
AC	Acceleration
BG	Begin Motion
CD	Contour Data
CM	Contour Mode
CS	Clear Motion Sequence
DC	Deceleration
DT	Contour Time Interval
EA	Select Master CAM axis
EB	Enable CAM mode
EG	Start CAM motion for slaves
EM	Define CAM cycles for each axis
EP	Define CAM table intervals & start point
EQ	Stop CAM motion for slaves
ES	Ellipse Scaling
ET	CAM table entries for slave axes
FE	Find Edge

FI	Find Index
GA	Master Axis for Gearing
GR	Gear Ratio
HM	Home
IP	Increment Position
JG	Jog Mode
LE	Linear Interpolation End
LI	Linear Interpolation Distance
LM	Linear Interpolation mode
LT	Latch Target
PA	Position Absolute
PR	Position Relative
SP	Speed
ST	Stop
VA	Vector acceleration
VD	Vector Deceleration
VE	Vector Sequence End
VM	Coordinated Motion Mode
VP	Vector Position
VR	Vector speed ratio
VS	Vector Speed

Program Flow

AD	After Distance
AI	After Input
AM	After Motion Complete
AP	After Absolute Position
AR	After Relative Distance
AS	At Speed
AT	After Time
AV	After Vector Distance
ELSE	ELSE Function for use with IF Conditional Statement
EN	End Program
ENDIF	End of IF Conditional Statement

HX	Halt Task
IF	IF Conditional Statement
IN	Input Variable
II	Input Interrupt
JP	Jump To Program Location
JS	Jump To Subroutine
MC	After motor is in position
MF	After motion -- forward direction
MG	Message
MR	After motion -- reverse direction
NO	No operation
RE	Return from Error Subroutine
RI	Return from Interrupt
TW	Timeout for in position
WC	Wait for Contour Data
WT	Wait
XQ	Execute Program
ZS	Zero Subroutine Stack

General Configuration

AF	Analog Feedback
AL	Arm Latch
BN	Burn
BP	Burn Program
BV	Burn Variables
CB	Clear Bit
CE	Configure Encoder
CN	Configure Switches
DA	De-Allocate Arrays
DE	Define Dual Encoder Position
DL	Download
DM	Dimension Arrays
DP	Define Position
EO	Echo Off
HS	Handle Switch
LC	Lock Controller

LS	List
MO	Motor Off
MT	Motor Type
MW	Modbus Wait
OB	Output Bit
OP	Output Port
PF	Position Format
PW	Password
QD	Download Array
QU	Upload Array
RA	Record Array
RC	Record
RD	Record Data
RS	Reset
SB	Set Bit
UL	Upload
VF	Variable Format

Control Filter Settings

DV	Damping for dual loop
FA	Acceleration Feed Forward
FV	Velocity Feed Forward
IL	Integrator Limit
IT	Smoothing Time Constant - Independent
KD	Derivative Constant
KI	Integrator Constant
KP	Proportional Constant
NB	Notch Bandwidth
NF	Notch Filter
NZ	Notch Zero
OF	Offset
PL	Pole Filter
SH	Servo Here
TL	Torque Limit
TM	Sample Time
VT	Smoothing Time Constant - Vector

Status

RP	Report Command Position
RL	Report Latch
SC	Stop Code
TB	Tell Status
TC	Tell Error Code
TD	Tell Dual Encoder
TE	Tell Error
TI	Tell Input
TP	Tell Position
TH	Tell Handle
RL	Report Latch
TR	Trace
TS	Tell Switches
TT	Tell Torque
TV	Tell Velocity

Error And Limits

BL	Reverse Software Limit
ER	Error Limit
FL	Forward Software Limit
OE	Off on Error

Arithmetic Functions

@ABS	Absolute Value
@ACOS	Arc Cosine
@AN	Return AnalogInput
@ASIN	Arc Sine
@ATAN	Arc Tangent
@COM	Return 2’s Complement
@COS	Cosine
@FRAC	Fraction Portion
@IN	Return Digital Input
@INT	Integer Portion
@OUT	Return Output
@RND	Round

@SIN	Sine
@SQR	Square root
@TAN	Tangent
+	Add
-	Subtract
*	Multiply
/	Divide
&	And
|	Or
( )	Parentheses

NOTES:

6 Programming Motion

Overview

The SMC–4000 provides several modes of motion, including independent positioning and jogging, coordinated motion, electronic cam motion, and electronic gearing. Each one of these modes is discussed in the following sections.

The SMC–4000 is a single axis controller and uses X-axis motion only. The example applications described below will help guide you to the appropriate mode of motion.

Example Application	Mode of Motion	Commands
Absolute or relative positioning where each axis is independent and follows prescribed velocity profile.	Independent Axis Positioning	PA,PR SP,AC,DC
Velocity control where no final endpoint is prescribed. Motion stops on Stop command.	Independent Jogging	JG AC,DC ST
Motion Path described as incremental position points versus time.	Contour Mode	CM CD DT WC
2, 3, or 4-axis coordinated motion where path is described by linear segments.	Linear Interpolation	LM LI,LE VS,VR VA,VD
Electronic gearing where slave axis is scaled to master axis which can move in both directions.	Electronic Gearing	GA GR
Master/slave where slave axes must follow a master such as conveyer speed.	Electronic Gearing	GA GR
Moving along arbitrary profiles or mathematically prescribed profiles such as sine or cosine trajectories.	Contour Mode	CM CD DT WC
Teaching or Record and Play Back	Contour Mode with Automatic Array Capture	CM CD DT WC RA RD RC
Backlash Correction	Dual Loop	DV
2-D motion path consisting of arc segments and linear segments, such as engraving or quilting.	Coordinated Motion	VM, CA VP CR VS,VR VA,VD VE

Following a trajectory based on a master encoder position	Electronic Cam	EA EM EP ET EB EG EQ
Smooth motion while operating in independent axis positioning	Independent Motion Smoothing	IT
Smooth motion while operating in vector or linear interpolation positioning	Vector Smoothing	VT

Independent Axis Positioning

In this mode, motion between the specified axes is independent, and each axis follows its own profile. The user specifies the desired absolute position (PA) or relative position (PR), slew speed (SP), acceleration ramp (AC), and deceleration ramp (DC), for each axis. On begin (BG), the SMC–4000 profiler generates the corresponding trapezoidal or triangular velocity profile and position trajectory. The controller determines a new command position along the trajectory every sample period until the specified profile is complete. Motion is complete when the last position command is calculated by the SMC–4000 profiler.

***Note The motor motion may not be complete when the profile has been completed, however, the next motion command may be specified.**

The BG (Begin) command can be issued for all axes either simultaneously or independently. X or Y axis specifiers are required to select the axes for motion. When no axes are specified, this causes motion to begin on all axes.

The SP (Speed) and AC (Acceleration) commands can be changed at any time during motion, however, the DC (Deceleration) and PR (Position Relative)/PA (Position Absolute) commands cannot be changed until motion is complete. Remember, motion is complete (AM (After Motion)) when the profiler is finished, not when the actual motor is in position. The ST (Stop) command can be issued at any time to decelerate the motor to a stop before it reaches its final position.

An IP (Increment Position) command may be specified during motion as long as the additional move is in the same direction. Here, the user specifies the desired position increment, n. The new target is equal to the old target plus the increment, n. Upon receiving the IP command, a revised profile will be generated for motion towards the new end position. The IP command does not require a begin. Note: If the motor is not moving, the IP command is equivalent to the PR (Position Relative) and BG (Begin) command combination.

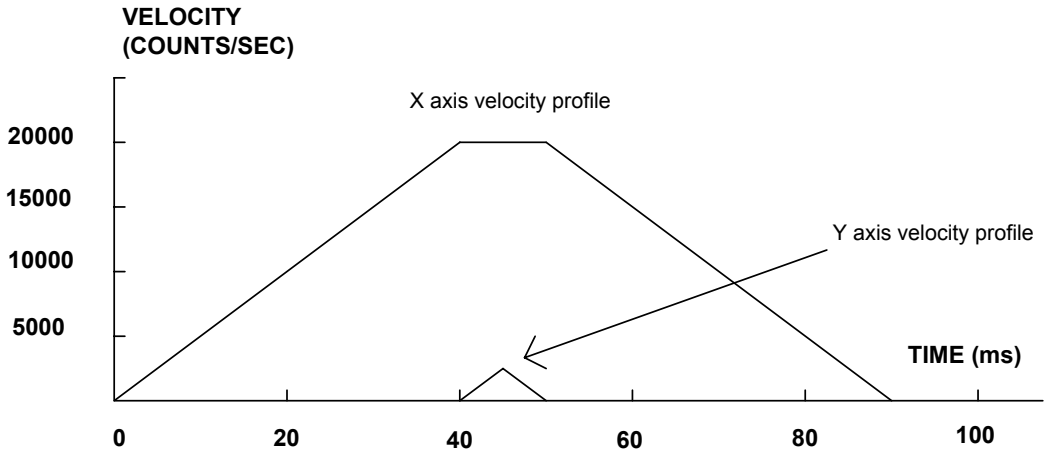
Command Summary - Independent Axis

Command	Description
PR x,y	Specifies relative distance
PA x,y	Specifies absolute position
SP x,y	Specifies slew speed
AC x,y	Specifies acceleration rate
DC x,y	Specifies deceleration rate
BG XY	Starts motion
ST XY	Stops motion before end of move
IP x,y	Changes position target
IT x,y	Time constant for independent motion smoothing
AM XY	Trippoint for profiler complete
MC XY	Trippoint for "in position"

The lower case specifiers (x,y) represent position values for each axis. The SMC–4000 also allows use of single axis specifiers such as PRY=2000.

The following illustration - *Velocity Profiles of XY* shows the velocity profiles for the X and Y axis.

Instruction	Interpretation
#A	Begin Program
PR 2000,100	Specify relative position movement of 2000 and 100 counts for the X and Y axes.
SP 20000,5000	Specify speed of 20000 and 5000 counts / sec
AC 500000,500000	Specify acceleration of 500000 counts / sec^2 for the X and Y axes
DC 500000,500000	Specify deceleration of 500000 counts / sec^2 for the X and Y axes
BG X	Begin motion on the X axis
WT 40	Wait 40 msec
BG Y	Begin motion on the Y axis
EN	End Program



Velocity Profiles of XY

Notes on *Velocity Profiles of XY* illustration: The X axis has a 'trapezoidal' velocity profile, while the Y axis has a 'triangular' velocity profile. The X axis accelerates to the specified speed, moves at this constant speed, and then decelerates such that the final position agrees with the commanded position, PR. The Y axis accelerates, but before the specified speed is achieved, must begin deceleration such that the axis will stop at the commanded position.

Independent Jogging

The jog mode of motion is very flexible because speed, direction and acceleration can be changed during motion. The user specifies theJG (Jog), AC (Acceleration), and DC (Deceleration) commands for each axis. The direction of motion is specified by the sign of the JG parameters. When the BG (Begin) command is given, the motor accelerates up to speed and continues to jog at that speed until a new speed or ST (Stop) command is issued. If the jog speed is changed during motion, the controller will make an accelerated (or decelerated) change to the new speed.

An instant change to the motor position can be made with the use of the IP (Increment Position) command. Upon receiving this command, the controller commands the motor to a position which is equal to the specified increment plus the current position. This command is useful when trying to synchronize the position of two motors while they are moving.

Note that the controller operates as a closed-loop position controller while in the jog mode. The SMC–4000 converts the velocity profile into a position trajectory and a new position target is generated every sample period. This method of control results in precise speed regulation with phase lock accuracy.

Command Summary - Jogging

Command	Description
AC x,y	Specifies acceleration rate
BG XY	Begins motion
DC x,y	Specifies deceleration rate
IP x,y	Increments position instantly
IT x,y	Time constant for independent motion smoothing
JG +/-x,y	Specifies jog speed and direction
ST XY	Stops motion

Parameters can be set with individual axis specifiers such as JGY=2000 (set jog speed for Y axis to 2000) or AC 400000, 400000 (set acceleration for X and Y axes to 400000).

Linear Interpolation Mode

The SMC–4000 provides a linear interpolation mode for 2 or more axes. In linear interpolation mode, motion is coordinated to maintain the prescribed vector speed, acceleration, and deceleration along the specified path. The motion path is described in terms of incremental distances. An unlimited number of incremental segments may be given in a continuous move sequence, making the linear interpolation mode ideal for following a piece-wise linear path. There is no limit to the total move length.

The LM (Linear Interpolation Mode) command selects the Linear Interpolation mode and axes for interpolation. For example, LM X selects the X axis for linear interpolation.

When using the linear interpolation mode, the LM command only needs to be specified once unless the axes for linear interpolation change.

Specifying the Coordinate Plane

The SMC–4000 allows for 2 separate sets of coordinated axes for linear interpolation mode or vector mode. These two sets are identified by the letters S and T.

To specify vector commands the coordinate plane must first be identified. This is done by issuing the command CAS to identify the S plane or CAT to identify the T plane. All vector commands will be applied to the active coordinate system until changed with the CA (Coordinate Axes) command.

Specifying Linear Segments

The command LI (Linear Interpolation Distance) x specifies the incremental move distance for each axis. This means motion is prescribed with respect to the current axis position. Up to 511 incremental move segments may be given prior to the Begin Sequence (BGS) command. Once motion has begun, additional LI segments may be sent to the controller.

The CS (Clear Sequence) command can be used to remove LI segments stored in the buffer prior to the start of the motion. To stop the motion, use the instructions STS or AB (Abort). The ST (Stop) command causes a decelerated stop. The command AB causes an instantaneous stop and aborts the program, and the command AB1 aborts the motion only.

The LE (Linear Interpolation End) command must be used to specify the end of a linear move sequence. This command tells the controller to decelerate to a stop following the last LI command. If an LE command is not given, an Abort AB1 must be used to abort the motion sequence.

It is the responsibility of the user to keep enough LI segments in the SMC–4000 sequence buffer to ensure continuous motion. If the controller receives no additional LI segments and no LE command, the controller will stop motion instantly at the last vector. There will be no controlled deceleration. LM? or _LM returns the available spaces for LI segments that can be sent to the buffer. 511 returned means the buffer is empty and 511 LI segments can be sent. A zero means the buffer is full and no additional segments can be sent. As long as the buffer is not full, additional LI segments can be sent.

The instruction _CS returns the number of the segment being processed. As the segments are processed, _CS increases, starting at zero. This function allows the host computer to determine which segment is being completed.

Additional Commands

The commands VS (Vector Speed) n, VA (Vector Acceleration) n, and VD (Vector Deceleration) n are used to specify the vector speed, acceleration, and deceleration. The SMC–4000 computes the vector speed based on the axes specified in the LM mode. For example, with LMXY, the vecor speed is computed as:

$$VS = \sqrt{SPx^2 + SPy^2}$$

VT (Vector Time Constant) is used to set the S-curve smoothing constant for coordinated moves. The command AV (After Vector Distance) n is the ‘After Vector’ trippoint, which halts program execution until the vector distance of n has been reached.

Specifying Vector Speed for Each Segment

The instruction VS (Vector Speed) has an immediate effect and, therefore, must be given at the required time. In some applications, such as CNC, it is necessary to attach various speeds to different motion segments. This can be done with two functions: < n and > m

For example:LI x < n >m

The first command, < n, is equivalent to commanding VSn at the start of the given segment and will cause an acceleration toward the new commanded speed, subject to the other constraints.

The second function, > m, requires the vector speed to reach the value m at the end of the segment. Note that the function > m may start the deceleration within the given segment or during previous segments, as needed to meet the final speed requirement, under the given values of VA (Vector Acceleration) and VD (Vector Deceleration).

Note, however, that the controller works with one > m command at a time. As a consequence, one function may be masked by another. For example, if the function >100000 is followed by >5000, and the distance for deceleration is not sufficient, the second condition will not be met. The controller will attempt to lower the speed to 5000.

As an example, consider the following program.

Instruction	Interpretation
#ALT	Label for alternative program
DP 0,0	Define Position of X and Y axis to be 0
LMX	Define linear mode.
LI 4000 <4000 >1000	Specify first linear segment with a vector speed of 4000 and end speed 1000
LI 1000 < 4000 >1000	Specify second linear segment with a vector speed of 4000 and end speed 1000
LI 0 < 4000 >1000	Specify third linear segment with a vector speed of 4000 and end speed 1000
LE	End linear segments
BGS	Begin motion sequence
EN	Program end

Changing Feedrate:

The command VR n allows the feedrate, VS, to be scaled between 0 and 10 with a resolution of 0.0001. This command takes effect immediately and causes VS to be scaled. VR also applies when the vector speed is specified with the '<' operator. This is a useful feature for feedrate override. VR does not ratio the accelerations. For example, VR .5 results in the specification VS 2000 to be divided in half.

Command Summary - Linear Interpolation

Command	Description
LM abcd	Specify axes for linear interpolation
LM?	Returns number of available spaces for linear segments in SMC–4000 sequence buffer. Zero means buffer full. 511 means buffer empty.
LI abcd < n	Specify incremental distances relative to current position, and assign vector speed n.
VS n	Specify vector speed
VA n	Specify vector acceleration
VD n	Specify vector deceleration
VR n	Specify the vector speed ratio
BGS	Begin Linear Sequence
CS	Clear sequence
LE	Linear End- Required at end of LI command sequence
MG_LE	Returns the length of the vector (resets after 2147483647)
AMS	Trippoint for After Sequence complete
AV n	Trippoint for After Relative Vector distance, n
VT	Motion smoothing constant for vector moves

To illustrate the ability to interrogate the motion status, consider the first motion segment of our example, #LMOVE, where the X axis moves toward the point X=5000. Suppose that when X=3000, the controller is interrogated using the command 'MG _AV'. The returned value will be 3000. The value of _CS and _VPX will be zero.

Example

Linear Interpolation Motion

In this example, the XY system is required to perform a 90° turn. In order to slow the speed around the corner, we use the AV 4000 trippoint, which slows the speed to 1000 count/s. Once the motors reach the corner, the speed is increased back to 4000 cts / s.

Instruction	Interpretation
#LMOVE	Label
DP 0,0	Define position of X andY axes to be 0.
LMXY	Define linear mode between X and Y axes.
LI 5000,0	Specify first linear segment
LI 0,5000	Specify second linear segment
LE	End linear segments
VS 4000	Specify vector speed
BGS	Begin motion sequence

AV 4000	Set trippoint to wait until vector distance of 4000 is reached.
VS 1000	Change vector speed
AV 5000	Set trippoint to wait until vector distance of 5000 is reached.
VS 4000	Change vector speed
EN	Program end

Linear Move

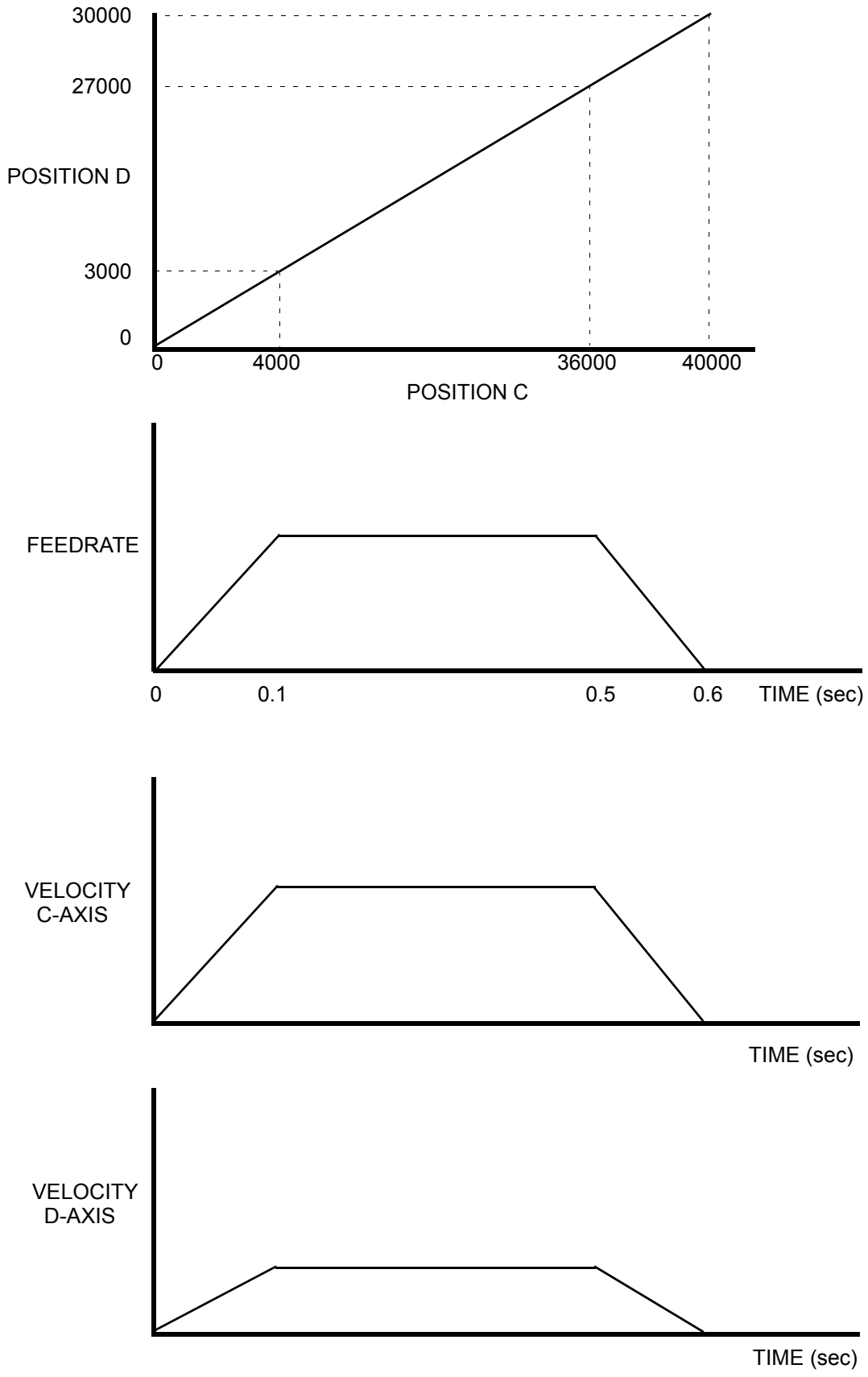
Make a coordinated linear move in the CD plane. Move to coordinates 40000,30000 counts at a vector speed of 100000 counts/sec and vector acceleration of 1000000 counts/sec^2.

Instruction	Interpretation
LM CD	Specify axes for linear interpolation
LI,,40000,30000	Specify CD distances
LE	Specify end move
VS 100000	Specify vector speed
VA 1000000	Specify vector acceleration
VD 1000000	Specify vector deceleration
BGS	Begin sequence

Note that the above program specifies the vector speed, VS, and not the actual axis speeds VC and VD. The axis speeds are determined by the SMC–4000 from:

$$VS = \sqrt{VC^2 + VD^2}$$

The resulting profile is shown in the figures on the following page.



LINEAR INTERPOLATION EXAMPLE

Multiple Moves

This example makes a coordinated linear move in the XY plane. The arrays VA and VB are used to store 750 incremental distances which are filled by the program #LOAD.

Instruction	Interpretation
#LOAD	Load Program
DM VA[750],VB[750]	Define Array
count=0	Initialize Counter
n=0	Initialize position increment
#LOOP	LOOP
VA [count]=n	Fill Array VA
VB [count]=n	Fill Array VB
n=n+10	Increment position
count=count+1	Increment counter
JP #LOOP, count <750	Loop if array not full
#A	Label
LM XY	Specify linear mode for AB
count=0	Initialize array counter
#LOOP2;JP#LOOP2,_LM=0	If sequence buffer full, wait
JS#C, count=500	Begin motion on 500th segment
LI VA[count],VB[count]	Specify linear segment
count=count+1	Increment array counter
JP #LOOP2, count <750	Repeat until array done
LE	End Linear Move
AMS	After Move sequence done
MG “DONE”	Send Message
EN	End program
#C;BGS;EN	Begin Motion Subroutine

Vector Mode: Linear and Circular Interpolation Motion

The SMC–4000 allows a long 2-D path consisting of linear and arc segments to be prescribed. Motion along the path is continuous at the precribed vector speed even at transitions between linear and circular segments. The SMC–4000 performs all of the complex computations of linear and circular interpolation, freeing the host PC from this time intensive task.

The coordinated motion mode is similar to the linear interpolation mode. Any pair of two axes may be selected for coordinated motion consisting of linear and circular segments. In addition, a third axis can be controlled such that it remains tangent to the motion of the selected pair of axes. Note that only one pair of axes can be specified for coordinated motion at any given time.

The command VM m,n,p where “m” and “n” are the coordinated pair and “p” is the tangent axis.

NOTE: The commas which separate m,n and p are not necessary. For example, VM XYZ selects the XY axes for coordinated motion and the Z-axis as the tangent.

Specifying the Coordinate Plane

The SMC–4000 allows for 2 separate sets of coordinated axes for linear interpolation mode or vector mode. These two sets are identified by the letters S and T.

To specify vector commands the coordinate plane must first be identified. This is done by issuing the command CAS to identify the S plane or CAT to identify the T plane. All vector commands will be applied to the active coordinate system until changed with the CA command.

Specifying Vector Segments

The motion segments are described by two commands: VP for linear segments and CR for circular segments. Once a set of linear segments and/or circular segments have been specified, the sequence is ended with the command VE. This defines a sequence of commands for coordinated motion. Immediately prior to the execution of the first coordinated movement, the controller defines the current position to be zero for all movements in a sequence.

***Note This ‘internal’ definition of zero does not affect the absolute coordinate system or subsequent coordinated motion sequences.**

The command, VP xy specifies the coordinates of the end points of the vector movement with respect to the starting point. Non-Sequential Axes do not require comma delimitation. The command, CR r,q,d defines a circular arc with a radius r, starting angle of q, and a traversed angle d. The notation for q is that zero corresponds to the positive horizontal direction, and for both q and d, the counterclockwise (CCW) rotation is positive.

Up to 511 segments of CR and VP may be specified in a single sequence and must be ended with the command VE. The motion can be initiated with a Begin Sequence (BGS) command. Once motion starts, additional segments may be added.

The Clear Sequence (CS) command can be used to remove previous VP and CR commands which were stored in the buffer prior to the start of the motion. To stop the motion, use the instructions STS or AB1. STS stops motion at the specified deceleration. AB1 aborts the motion instantaneously.

The Vector End (VE) command must be used to specify the end of the coordinated motion. This command tells the controller to decelerate to a stop following the last motion in the sequence. If a VE command is not given, an Abort (AB1) must be used to abort the coordinated motion sequence.

The user must keep enough motion segments in the SMC–4000 sequence buffer to ensure continuous motion. If the controller receives no additional motion segments and no VE command, the controller will stop motion instantly at the last vector. There will be no controlled deceleration. LM? or _LM returns the available spaces for motion segments that can be sent to the buffer. 511 returned means the buffer is empty and 511 segments can be sent. A zero means the buffer is full and no additional segments can be sent. As long as the buffer is not full, additional segments can be sent at the PCI bus speeds.

The operand _CS can be used to determine the value of the segment counter.

Additional Commands

The commands VS n, VA n and VD n are used for specifying the vector speed, acceleration, and deceleration. VT is the motion smoothing constant used for coordinated motion.

Specifying Vector Speed for Each Segment:

The vector speed may be specified by the immediate command VS. It can also be attached to a motion segment with the instructions

VP x < n >m

The first parameter, <n, is equivalent to commanding VSn at the start of the given segment and will cause an acceleration toward the new commanded speeds, subject to the other constraints.

The second parameter, > m, requires the vector speed to reach the value m at the end of the segment. Note that the function > m may start the deceleration within the given segment or during previous segments, as needed to meet the final speed requirement, under the given values of VA and VD.

Note, however, that the controller works with one > m command at a time. As a consequence, one function may be masked by another. For example, if the function >100000 is followed by >5000, and the distance for deceleration is not sufficient, the second condition will not be met. The controller will attempt to lower the speed to 5000, but will reach that at a different point.

Changing Feedrate:

The command VR n allows the feedrate, VS, to be scaled from 0 to 10 times with a resolution of .0001. This command takes effect immediately and causes VS to be scaled. VR also applies when the vector speed is specified with the '<' operator. This is a useful feature for feedrate override. VR does not ratio the accelerations. For example, VR .5 results in the specification VS 2000 to act as VS 1000.

Trippoints:

The AV n command is the After Vector trippoint, which waits for the vector relative distance of n to occur before executing the next command in a program.

Tangent Motion:

Several applications, such as cutting, require a third axis (i.e. a knife blade), to remain tangent to the coordinated motion path. To handle these applications, the SMC–4000 allows one axis to be specified as the tangent axis. The VM command provides parameter specifications for describing the coordinated axes and the tangent axis.

V,M m,n,p	m,n specifies coordinated axes, p specifies tangent axis such as A,B,C or D p=N turns off tangent axis

Before the tangent mode can operate, it is necessary to assign an axis via the TN m,n command. m defines the scale factor in counts/degree and n defines the tangent position that equals zero degrees in the coordinated motion plane. The operand _TN can be used to return the initial position of the tangent axis.

Command Summary - Coordinated Motion Sequence

Command	Description
VM m,n,p	Specifies the axes for planar motion where m and n represent the planar axes and p is the tangent axis.
VP m,n	Return coordinate of last point, where m=X,Y,Z or W.
VS n	Specify vector speed or feedrate of sequence.
VA n	Specify vector acceleration along the sequence.
VD n	Specify vector deceleration along the sequence.
VR n	Specify vector speed ratio
BGS	Begin motion sequence
CS	Clear sequence.
AV n	Trippoint for After Relative Vector distance, n.
AMS	Holds execution of next command until Motion Sequence is complete.
TN m,n	Tangent scale and offset
ES m,n	Ellipse scale factor
VT	S curve smoothing constant for coordinated moves

Operand Summary - Coordinated Motion Sequence

Operand	Description
_vpm	The absolute coordinate of the axes at the last intersection along the sequence.
_AV	Distance traveled.
_LM	Number of available spaces for linear and circular segments in the SMC–4000 sequence buffer. Zero means buffer is full. 512 means buffer is empty.
_CS	Segment counter - Number of the segment in the sequence, starting at zero.
_VE	Vector length of coordinated move sequence.

When AV is used as an operand, _AV returns the distance traveled along the sequence.

The operands _VPX and _VPY can be used to return the coordinates of the last point specified along the path.

Example

Tangent Axis

Assume an AB table with the C-axis controlling a knife. The C-axis has a 2000 quad counts/rev encoder and has been initialized after power-up to point the knife in the +B direction. A 180° circular cut is desired, with a radius of 3000, center at the origin and a starting point at (3000.0). The motion is CCW, ending at (-3000.0). Note that the 0° position in the AB plane is in the +A direction. This corresponds to the position -500 in the Z-axis, and defines the offset. The motion has two parts. First, A, B and C are driven to the starting point, and later, the cut is performed. Assume that the knife is engaged with output bit 0.

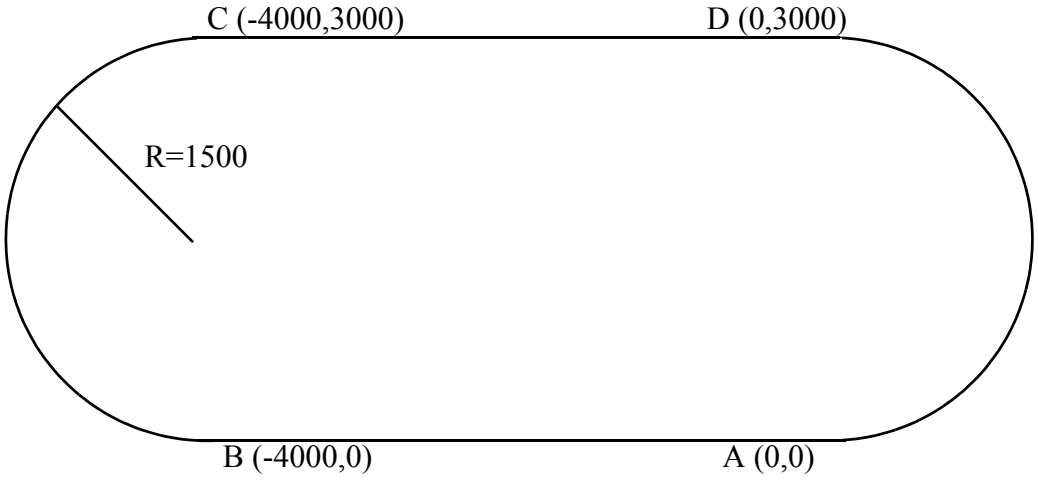
Instruction	Interpretation
#EXAMPLE	Example Program
VM ABC	AB coordinate with C as tangent
TN 2000/360,-500	2000/360 counts/degree, position -500 is 0 degrees in AB plane
CR 3000,0,180	3000 count radius, start at 0 and go to 180 CCW
VE	End vector
CB0	Disengage knife
PA 3000,0,_TN	Move A and B to starting position, move C to initial tangent position
BG ABC	Start the move to get into position
AM ABC	When the move is complete
SB0	Engage knife
WT50	Wait 50msec for the knife to engage
BGS	Do the circular cut
AMS	After the coordinated move is complete
CB0	Disengage knife
MG “ALL DONE”	
EN	End program

Coordinated Motion

Traverse the path shown in the following figure. Feedrate is 20000 counts/sec. Plane of motion is AB.

Instruction	Interpretation
VM AB	Specify motion plane
VS 20000	Specify vector speed
VA 1000000	Specify vector acceleration
VD 1000000	Specify vector deceleration
VP -4000,0	Segment AB
CR 1500,270,-180	Segment BC
VP 0,3000	Segment CD
CR 1500,90,-180	Segment DA

VE	End of sequence
BGS	Begin sequence



The Required Path

The resulting motion starts at the point A and moves toward points B,C,D,A. Suppose that we interrogate the controller when the motion is halfway between the points A and B.

The value of _AV is 2000

The value of _CS is 0

_VPA and _VPB contain the absolute coordinate of the point A

Suppose that the interrogation is repeated at a point, halfway between the points C and D.

The value of _AV is 4000+1500p+2000=10,712

The value of _CS is 2

_VPA,_VPB contain the coordinates of the point C

Electronic Gearing

This mode allows any axes to be electronically geared to some master axes. The masters may rotate in both directions and the geared axis will follow at the specified gear ratio. The gear ratio may be different for each axis and changed during motion.

The command GA (Master Axis for Gearing) ABCD specifies the master axes. GR (Gear Ratio) x,y specifies the gear ratios for the slaves where the ratio may be a number between +/-127.9999 with a fractional resolution of 0.0001. There are two modes: standard gearing and gantry mode. The gantry mode is enabled with the command GM. GR 0,0 turns off gearing in both modes. A limit switch or ST (Stop) command disables gearing in the standard mode but not in the gantry mode.

The command GM (Gantry Mode) a,b,c,d selects the axis to be controlled under the gantry mode. The parameter 1 enables gantry mode, and 0 disables it.

GR (Gear Ratio) causes the specified axes to be geared to the actual position of the master. The master axis is commanded with motion commands such as PR (Position Relative), PA (Position Absolute), or JG (Jog).

When the master axis is driven by the controller in the jog mode or an independent motion mode, it is possible to define the master as the command position of that axis, rather than the actual position. The designation of the commanded position master is the letter, C. For example, GACA indicates that the gearing is the commanded position of A.

An alternative gearing method is to synchronize the slave motor to the commanded vector motion of several axes performed by GAS. For example, if the A and B motor form a circular motion, the C axis may move in proportion to the vector move. Similarly, if A, B and C perform a linear interpolation move, W can be geared to the vector move.

Electronic gearing allows the geared motor to perform a second independent or coordinated move in addition to the gearing. For example, when a geared motor follows a master at a ratio of 1:1, it may be advanced an additional distance with PR (Position Relative), JG (Jog), VP (Vector Position), or LI (Linear Interpolation Distance) commands.

Command Summary - Electronic Gearing

Command	Description
GA n	Specifies master axes for gearing where n=A,B,C,D for main encoder as master.
	n=CA,CB,CC,CD for commanded position.
	n=DA,DB,DC,DD for auxiliary encoders.
	n=S or T for gearing to coordinated motion.
GR a,b,c,d	Sets gear ratio for slave axes. 0 disables electronic gearing for specified axis.
GM a,b,c,d	1 sets gantry mode, 0 disables gantry mode.
MR a,b,c,d	Trippoint for reverse motion past specified value. Only one field may be used.
MF a,b,c,d	Trippoint for forward motion past specified value. Only one field may be used.

Example

Simple Master Slave

Master axis moves 10000 counts at slew speed of 100000 counts/sec. B is defined as the master. A,C,D are geared to master at ratios of 5,-.5 and 10 repectively.

Instruction	Interpretation
GA B,,B,B	Specify master axis as B
GR 5,,-.5,10	Set gear ratios
PR ,10000	Specify B position
SP ,100000	Specify B speed
BGB	Begin motion

Electronic Gearing

Objective: Run two geared motors at speeds of 1,132 and -0.045 times the speed of an external master. The master is driven at speeds between 0 and 1800 RPM (2000 counts/rev encoder).

Solution: Use a SMC–4000 controller, where the C-axis is the master and A and B are the geared axes.

Instruction	Interpretation
MO C	Turn C off, for external master
GA C,C	Set C as the master axis for both A and B
GR 1.132,-.045	Specify gear ratios

Now suppose the gear ratio of the A-axis is to change on-the-fly to 2. This can be achieved by commanding:

GR 2 Specify gear ratio for A axis to be 2

Gantry Mode

In applications where both the master and the follower are controlled by the SMC–4000 controller, it may be desired to synchronize the follower with the commanded position of the master, rather than the actual position. This eliminates the coupling between the axes which may lead to oscillations.

For example, assume that a gantry is driven by two axes, A,B on both sides. This requires the gantry mode for strong coupling between the motors. The A-axis is the master and the B-axis the follower. To synchronize B with the commanded position of A, use the instructions:

Instruction	Interpretation
GA CA	Specify the commanded position of A as master for B
GR,1	Set gear ratio for Y as 1:1
GM,1	Set gantry mode
PR 3000	Command A motion
BG A	Start motion on A axis

You may also perform profiled position corrections in the electronic gearing mode. Suppose, for example, that you need to advance the slave 10 counts. Simply command:

IP ,10 Specify an incremental position movement of 10 on B axis.

Under these conditions, this IP command is equivalent to:

PR,10 Specify position relative movement of 10 on B axis

BGB Begin motion on B axis

Often the correction is quite large. Such requirements are common when synchronizing cutting knives or conveyor belts.

Synchronize two conveyor belts with trapezoidal velocity correction:

Instruction	Interpretation
GA,A	Define A as the master axis for B.
GR,2	Set gear ratio 2:1 for B
PR,300	Specify correction distance
SP,5000	Specify correction speed
AC,100000	Specify correction acceleration
DC,100000	Specify correction deceleration
BGB	Start correction

Electronic Cam

The electronic cam is a motion control mode which enables the periodic synchronization of the servo motor with an external device. Up to 4 axes can be slaved to one master axis. The master axis encoder must be input through a main encoder port.

The electronic cam is a more detailed type of electronic gearing which allows a table-based relationship between the axes. It allows synchronizing all the controller axes. For example, the SMC–4000 controller may have one master and up to four slaves (using a virtual master).

To illustrate the procedure of setting the cam mode, consider the cam relationship for the slave axis B, when the master is A. Such a graphic relationship is shown in the following illustration - *Electronic Cam Example*.

Step 1. Selecting the master axis

The first step in the electronic cam mode is to select the master axis. This is done with the instruction:

EAp where p=A,B,C,D and p is the selected master axis

For the given example, since the master is x, we specify EAA.

Step 2. Specify the master cycle and the change in the slave axis.

In the electronic cam mode, the position of the master is always expressed within one cycle. In this example, the position of x is always expressed in the range between 0 and 6000. Similarly, the slave position is also redefined such that it starts at zero and ends at 1500. At the end of a cycle when the master is 6000 and the slave is 1500, the positions of both x and y are redefined as zero. To specify the master cycle and the slave cycle change, use the instruction EM.

EM x;

where EMx specifies the cycle of the axis.

The cycle of the master is limited to 8,388,607 whereas the slave change per cycle is limited to 2,147,483,647. If the change is a negative number, the absolute value is specified. For the given example, the cycle of the master is 6000 counts and the change in the slave is 1500. Therefore, we use the instructions:

EM 1500

Step 3. Specify the master interval and starting point.

Next we need to construct the ECAM table. The table is specified at uniform intervals of master positions. Up to 1024 intervals are allowed. The size of the master interval and the starting point are specified by the instruction:

EP m,n

where m is the interval width in counts, and n is the starting point.

For the given example, we can specify the table by specifying the position at the master points of 0, 2000, 4000 and 6000. We can specify that by

EP 2000,0

Step 4. Specify the slave positions.

Next, we specify the slave positions with the instruction

ET[n]=x

where n indicates the order of the point.

The value, n, starts at zero and may go up to 1024. The parameter x indicates the corresponding slave position. For this example, the table may be specified by

ET[0]=0

ET[1]=3000

ET[2]=2250

ET[3]=1500

This specifies the ECAM table.

Step 5. Enable the ECAM

To enable the ECAM mode, use the command

EB n

where n=1 enables ECAM mode and n=0 disables ECAM mode.

Step 6. Engage the slave motion

To engage the slave motion, use the instruction

EG x

where x is the master positions at which the corresponding slaves must be engaged.

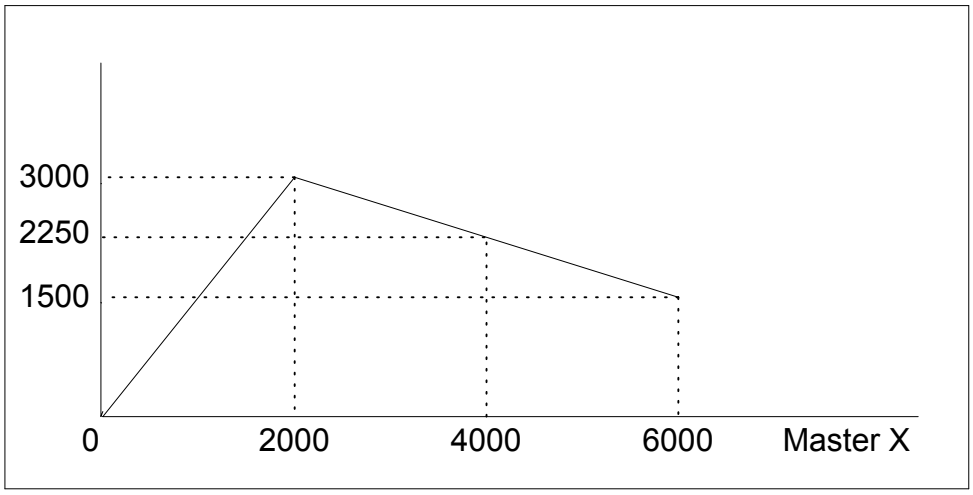
If the value of any parameter is outside the range of one cycle, the cam engages immediately. When the cam is engaged, the slave position is redefined, modulo one cycle.

Step 7. Disengage the slave motion

To disengage the cam, use the command

EQ x

where x is the master positions at which the corresponding slave axes are disengaged.



Electronic Cam Example

This disengages the slave axis at a specified master position. If the parameter is outside the master cycle, the stopping is instantaneous.

To illustrate the complete process, consider the cam relationship described by

the equation:

$Y = 0.5 * X + 100 \sin (0.18*X)$

where X is the master, with a cycle of 2000 counts.

The cam table can be constructed manually, point by point, or automatically by a program. The following program includes the set-up.

The instruction EAX defines X as the master axis. The cycle of the master is 2000. Over that cycle, X varies by 1000. This leads to the instruction EM 2000,1000.

Suppose we want to define a table with 100 segments. This implies increments of 20 counts each. If the master points are to start at zero, the required instruction is EP 20,0.

The following routine computes the table points. As the phase equals 0.18X and X varies in increments of 20, the phase varies by increments of 3.6°. The program then computes the values of SLAVE according to the equation and assigns the values to the table with the instruction ET[N] = SLAVE.

Instruction	Interpretation
#SETUP	Label
EAX	Select X as master
EM 2000, 1000	Specify slave cycle
EP 20,0	Master position increments
N = 0	Index
#LOOP	Loop to construct table from equation
P = N*3.6	Note 3.6 = 0.18*20
S = @SIN [P]*100	Define sine position
SLAVE = N*10+S	Define slave position
ET [N] = SLAVE	Define table
N = N+1	
JP #LOOP, N<=100	Repeat the process
EN	

Now suppose that the slave axis is engaged with a start signal, input 1, but that both the engagement and disengagement points must be done at the center of the cycle: X = 1000 and Y = 500. This implies that Y must be driven to that point to avoid a jump.

This is done with the program:

Instruction	Interpretation

#RUN	Label
EB1	Enable cam
PA,500	Y starting position
SP,5000	Y speed
BGY	Move Y motor
AM	After Y moved
AI1	Wait for start signal
EG,1000	Engage slave
AI – 1	Wait for stop signal
EQ,1000	Disengage slave
EN	End

Virtual Axis

The SMC–4000 controller has an additional virtual axis designated as the N axis. The axis has no encoder and no DAC. However, it can be commanded by the commands:

AC, DC, JG, SP, PR, PA, BG, IT, GA, VM, VP, CR, ST, DP, RP, EA.

The main use of the virtual axis is to serve as a virtual master in ECAM modes, and to perform an unnecessary part of a vector mode. These applications are illustrated by the following examples.

Ecam Master Example

Suppose that the motion of the AB axes is constrained along a path that can be described by an electronic cam table. Further assume that the ecam master is not an external encoder but has to be a controlled variable.

This can be achieved by defining the N axis as the master with the command EAN and setting the modulo of the master with a command such as EMN=4000. Next, the table is constructed. To move the constrained axes, simply command the N axis in the jog mode or with the PR and PA commands. For example:

PAN=2000

BGN

will cause the AB axes to move to the corresponding points on the motion cycle.

Sinusoidal Motion Example

The x axis must perform a sinusoidal motion of 10 cycles with an amplitude of 1000 counts and a frequency of 20Hz. This can be performed by commanding the A and N axes to perform circular motion. Note that the value of VS must be:

VS=2p * R * F

where R is the radius, or amplitude and F is the frequency in Hz.

Set VA and VD to maximum values for the fastest acceleration.

Instruction	**Interpretation**

VMAN	Select Axes
VA 68000000	Maximum Acceleration
VD 68000000	Maximum Deceleration
VS 125664	VS for 20Hz
CR 1000, -90, 3600	Ten cycles
VE	
BGS	

Contour Mode

The SMC–4000 also provides a contouring mode. This mode allows any arbitrary position curve to be prescribed for 1-4 axes. This is ideal for following computer generated paths such as parabolic, spherical or user-defined profiles. The path is not limited to straight line and arc segments and the path length may be infinite.

Specifying Contour Segments

The Contour Mode is specified with the command, CM, i.e.; CMX specifies contouring on the X axis.

A contour is described by position increments which are described with the command, CD x over a time interval, DT n. The parameter, n, specifies the time interval. The time interval is defined as 2^n ms, where n is a number between 1 and 8. The controller performs linear interpolation between the specified increments, where one point is generated for each millisecond.

Consider, for example, the following illustration - *The Required Trajectory.* The position X may be described by the points:

Point 1	X=0 at T=0ms
Point 2	X=48 at T=4ms
Point 3	X=288 at T=12ms
Point 4	X=336 at T=28ms

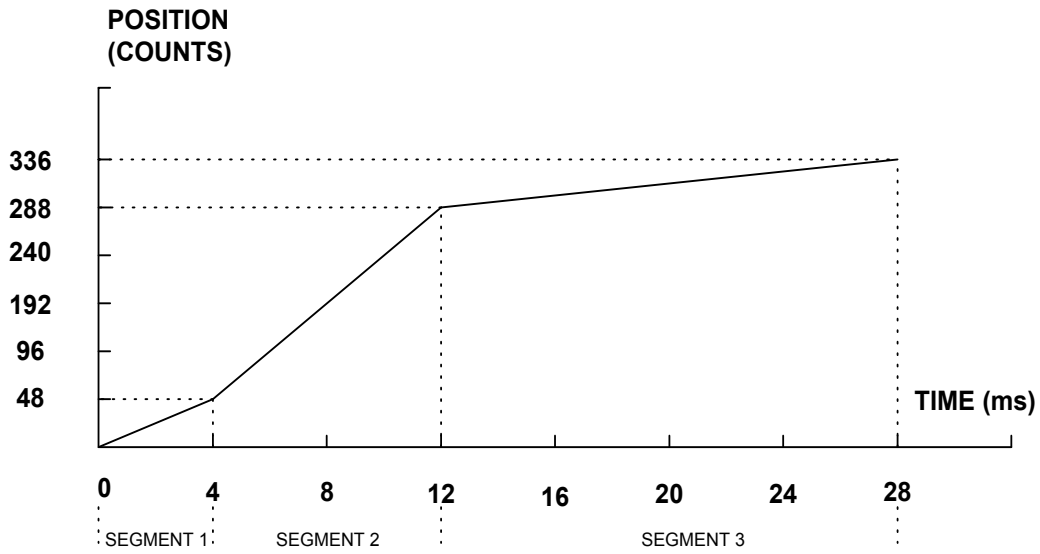
The same trajectory may be represented by the increments

Increment 1	DX=48	Time Increment =4	DT=2
Increment 2	DX=240	Time Increment =8	DT=3
Increment 3	DX=48	Time Increment =16	DT=4

When the controller receives the command to generate a trajectory along these points, it interpolates linearly between the points. The resulting interpolated points include the position 12 at 1 msec, position 24 at 2 msec, etc.

The programmed commands to specify the above example are:

Instruction	**Description**
#A	Label
CMX	Specifies X axis for contour mode
DT 2	Specifies first time interval, 2^2 ms
CD 48;WC	Specifies first position increment
DT 3	Specifies second time interval, 2^3 ms
CD 240;WC	Specifies second position increment
DT 4	Specifies the third time interval, 2^4 ms
CD 48;WC	Specifies the third position increment
DT0;CD0	Exits contour mode
EN	



The Required Trajectory

Additional Commands

The command, WC, is used as a trippoint "When Complete" or “Wait for Contour Data”. This allows the SMC–4000 to use the next increment only when it is finished with the previous one. Zero parameters for DT followed by zero parameters for CD exit the contour mode.

If no new data record is found and the controller is still in the contour mode, the controller waits for new data. No new motion commands are generated while waiting. If bad data is received, the controller responds with a ?.

Command Summary - Contour Mode

Command	Description
CM X	Specifies the X-axis for contouring mode. In a distributed control system, any non-contouring axes may be operated in other modes.
CD x	Specifies position increment over time interval. Range is +/-32,000. Zero ends contour mode.
DT n	Specifies time interval 2^n msec for position increment, where n is an integer between 1 and 8. Zero ends contour mode. If n does not change, it does not need to be specified with each CD.
WC	Waits for previous time interval to be complete before next data record is processed.

General Velocity Profiles

The Contour Mode is ideal for generating an arbitrary velocity profile. The velocity profile can be specified as a mathematical function or as a collection of points.

The design includes two parts: Generating an array with data points and running the program.

Motion Smoothing

The SMC–4000 controller allows the smoothing of the velocity profile to reduce mechanical vibrations in the system.

Trapezoidal velocity profiles have acceleration rates which change abruptly from zero to maximum value. The discontinuous acceleration results in jerk which causes vibration. The smoothing of the acceleration profile leads to a continuous acceleration profile and reduces the mechanical shock and vibration.

Using the IT (Independent Time Constant) and VT (Vector Time Constant) Commands (S curve profiling):

When operating with servo motors, motion smoothing can be accomplished with the IT and VT commands. These commands filter the acceleration and deceleration functions to produce a smooth velocity profile. The resulting velocity profile, known as an S curve, has continuous acceleration and results in reduced mechanical vibrations.

The smoothing function is specified by the following commands:

Command	Description
IT x,y	Independent time constant
VT n	Vector time constant

The command IT is used for smoothing independent moves of the type JG, PR, PA and the command VT is used to smooth vector moves of the type VM and LM.

The smoothing parameters x,y and n are numbers between 0 and 1 and determine the degree of filtering. The maximum value of 1 implies no filtering, resulting in trapezoidal velocity profiles. Smaller values of the smoothing parameters imply heavier filtering and smoother moves.

Note that the smoothing process results in longer motion time.

Stepper Motor Operation

When confirgured for stepper motor operation, several commands are interpreted differently than from servo mode. The following describes operation with stepper motors. Jumpers must be internally installed at position CN5 on the power board.

Specifying Stepper Motor Operation

In order to command stepper motor operation, the appropriate stepper mode jumpers must be installed. Stepper motor operation is specified by the command MT. The argument for MT is as follows:

2 Specifies a stepper motor with active low step output pulses.

-2 Specifies a stepper motor with active high step output pulses.

2.5 Specifies a stepper motor with active low step output pulses and reversed direction.

-2.5 Specifies a stepper motor with active high step output pulses and reversed direction.

Stepper Motor Smoothing

The KS (Step Motor Smoothing) command provides stepper motor smoothing. The effect of the smoothing can be thought of as a simple Resistor-Capacitor (single pole) filter. The filter occurs after the motion profiler and has the effect of smoothing out the spacing of pulses for a more smooth operation of the stepper motor. Use of KS is most applicable when operating in full step or half step operation. KS will cause the step pulses to be delayed in accordance with the time constant specified.

When operating with stepper motors, you will always have some amount of stepper motor smoothing, KS. Since the filtering effect occurs after the profiler, the profiler may be ready for additional moves before all of the step pulses have gone through the filter. It is important to consider this effect since steps may be lost if the controller is commanded to generate an additional move before the previous move has been completed. See the discussion below, *Monitoring Generated Pulses vs. Commanded Pulses*.

The general motion smoothing command, IT, can also be used. The purpose of the command, IT, is to smooth out the motion profile and decrease "jerk" due to acceleration.

Monitoring Generated Pulses vs. Commanded Pulses

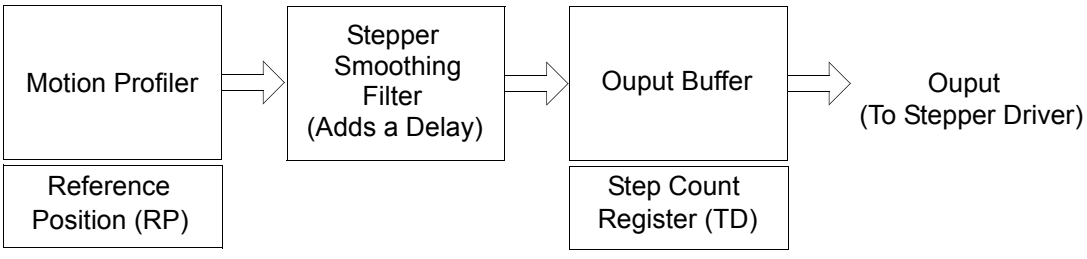
For proper controller operation, it is necessary to make sure that the controller has completed generating all step pulses before making additional moves. This is most particularly important if you are moving back and forth. For example, when operating with servo motors, the trippoint AM (After Motion) is used to determine when the motion profiler is complete and is prepared to execute a new motion command. However, when operating in stepper mode, the controller may still be generating step pulses when the motion profiler is complete. This is caused by the stepper motor smoothing filter, KS. To understand this, consider the steps the controller executes to generate step pulses:

First, the controller generates a motion profile in accordance with the motion commands.

Second, the profiler generates pulses as prescribed by the motion profile. The pulses that are generated by the motion profiler can be monitored by the command, RP (Reference Position). RP gives the absolute value of the position as determined by the motion profiler. The command, DP, can be used to set the value of the reference position. For example, DP 0, defines the reference position of the A axis to be zero.

Third, the output of the motion profiler is filtered by the stepper smoothing filter. This filter adds a delay in the output of the stepper motor pulses. The amount of delay depends on the parameter which is specified by the command, KS. As mentioned earlier, there will always be some amount of stepper motor smoothing. The default value for KS is 2 which corresponds to a time constant of 6 sample periods.

Fourth, the output of the stepper smoothing filter is buffered and is available for input to the stepper motor driver. The pulses which are generated by the smoothing filter can be monitored by the command, TD (Tell Dual). TD gives the absolute value of the position as determined by actual output of the buffer. The command, DP sets the value of the step count register as well as the value of the reference position. For example, DP 0 defines the reference position of the A axis to be zero.



Velocity Profiles of ABC

Motion Complete Trippoint

When used is stepper mode, the MC command will hold up execution of the proceeding commands until the controller has generated the same number of steps out of the step count register as specified in the commanded position. The MC trippoint (Motion Complete) is generally more useful than AM trippoint (After Motion) since the step pulses can be delayed from the commanded position due to stepper motor smoothing.

Using an Encoder with Stepper Motors

An encoder may be used on a stepper motor to check the actual motor position with the commanded position. If an encoder is used, it must be connected to the main encoder input.

NOTE: The auxiliary encoder is not available while operating with stepper motors. The position of the encoder can be interrogated by using the command, TP. The position value can be defined by using the command, DE.

NOTE: Closed loop operation with a stepper motor is not possible.

Command Summary - Stepper Motor Operation

Command	Description
DE	Define Encoder Position (When using an encoder)
DP	Define Reference Position and Step Count Register
IT	Motion Profile Smoothing - Independent Time Constant
KS	Stepper Motor Smoothing
MT	Motor Type (2, -2, 2.5, or -2.5 for stepper motors)
RP	Report Commanded Position
TD	Report number of step pulses generated by controller
TP	Tell Position of Encoder

Operand Summary - Stepper Motor Operation

Operand	Description
_DEa	Contains the value of the step count register for the “a” axis
_DPa	Contains the value of the main encoder for the “a” axis
_ITa	Contains the value of the Independent Time constant for the “a” axis
_KSa	Contains the value of the Stepper Motor Smoothing Constant for the “a” axis
_MTn	Contains the motor type value for the “a” axis
_RPn	Contains the commanded position generated by the profiler for the “a” axis
_TDa	Contains the value of the step count register for the “a” axis
_TPa	Contains the value of the main encoder for the “a” axis

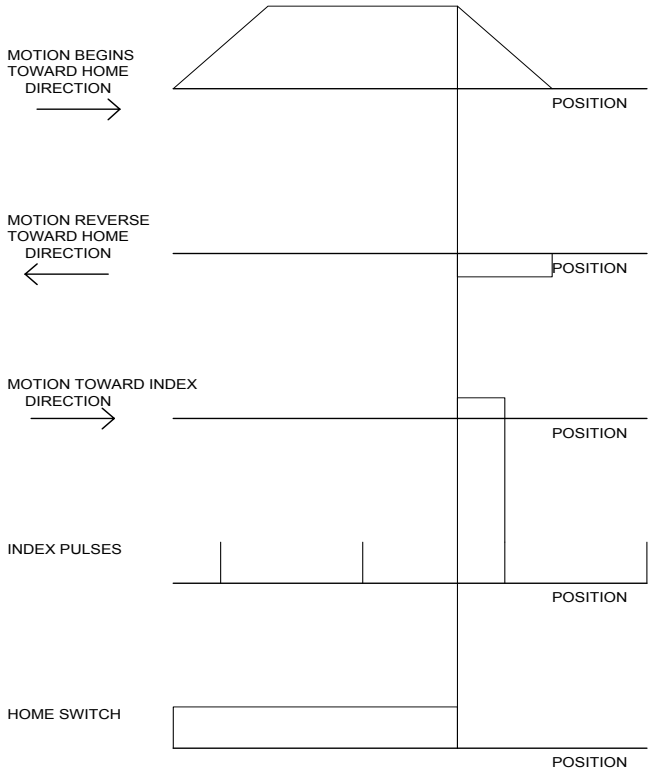
Homing

The Find Edge (FE) and Home (HM) instructions are used to home the motor to a mechanical reference. This reference is connected to the Home input line. The HM command initializes the motor to the encoder index pulse in addition to the Home input. The configure command (CN) defines polarity of the home input.

The Find Edge (FE) instruction is useful for initializing the motor to a home switch. The home switch is connected to the Home input. When the Find Edge command and Begin are used, the motor will accelerate up to the slew speed and slew until a transition is detected on the homing line. The motor will then decelerate to a stop. A high deceleration value must be input before the find edge command is issued for the motor to decelerate rapidly after sensing the home switch. The velocity profile generated is shown in the following illustration - *Motion intervals in the Home sequence.*

The Home (HM) command can be used to position the motor on the index pulse after the home switch is detected. This allows for finer positioning on initialization. The command sequence HM and BG causes the following sequence of events to occur.

1. Upon beginning, the motor accelerates to the slew speed. The direction of its motion is determined by the homing input. A zero (GND) will cause the motor to start in the forward direction; +24V will cause it to start in the reverse direction. The CN command defines the polarity of the home input.
2. Upon detecting a change in state on the home input, the motor begins decelerating to a stop.
3. The motor then traverses very slowly back until the home switch toggles again.
4. The motor then traverses forward until the encoder index pulse is detected.
5. The SMC–4000 defines the home position as the position at which the index was detected and sets the encoder reading at this position to zero.



Motion intervals in the Home sequence

High Speed Position Capture (Latch Function)

Often it is desirable to capture the position precisely for registration applications. The SMC–4000 provides a position latch feature. This feature allows position of the main X–axis to be captured within 25 microseconds of an external low input signal. General input 1 is the corresponding latch input for the main (X–axis) encoder.

***Note To insure a position capture within 25 microseconds, the input signal must be a transition from high to low.**

The SMC–4000 software commands AL and RL are used to arm the latch and report the latched position. The steps to use the latch are as follows:

1. Give the AL X command to arm the latch for the main (X–axis) encoder.
2. Test to see if the latch has occurred by using the _ALX command. Example, V1=_ALX returns the state of the X latch to the variable V1. V1 is 1 if the latch has not occurred.
3. After the latch has occurred, read the captured position with the RLX command or _RLX.

***Note The latch must be re-armed after each latching event.**

NOTES

7 Application Programming

Introduction

The SMC–4000 programming language is a powerful language that allows users to customize a program to handle their application. Complex programs can be downloaded into the SMC–4000 memory for later execution. Utilizing the SMC–4000 to execute sophisticated programs frees the host computer for other tasks. The host computer can still send commands to the controller any time, even while a program is being executed.

In addition to standard motion commands, the SMC–4000 provides commands that allow the SMC–4000 to make its own decisions. These commands include conditional jumps, event triggers, and subroutines. For example, the command JP#LOOP, N<10 causes a jump to the label #LOOP if the variable N is less than 10.

For flexibility, the SMC–4000 provides 510 user-defined variables, arrays and arithmetic functions, i.e.; length in a cut-to-length operation can be specified as a variable in a program and assigned by an operator.

The following sections in this chapter discuss all aspects of creating applications programs.

Program Format

An SMC–4000 program consists of several SMC–4000 instructions combined to solve a machine control application. Action instructions, such as starting and stopping motion, are combined with Program Flow instructions to form the complete program. Program Flow instructions evaluate real-time conditions, such as elapsed time or motion complete, and alter program flow accordingly.

A delimiter must separate each SMC–4000 instruction in a program. Valid delimiters are the semicolon (;) or carriage return. The semicolon is used to separate multiple instructions on a single program line. A carriage return enters the final command on a program line.

All SMC–4000 programs must begin with a label and end with an End (EN) statement. Labels start with the pound (#) sign followed by a maximum of seven characters. The first character must be a letter; after that, numbers are permitted. Spaces are not permitted.

The maximum number of labels that may be defined is 126.

Valid labels

#BEGIN

#SQUARE

#X1

#BEGIN1

Invalid labels

#1Square

#123

Special Labels

There are also some special labels, which are used to define input interrupt subroutines, limit switch subroutines, error handling subroutines, and command error subroutines. Special labels provide the application program a method of handling situations that would otherwise be difficult to program.

#AUTO	Label for automatic program start
#AUTOERR	Execute power-up if checksum error
#CMDERR	Label for incorrect command subroutine
#ININT	Label for Input Interrupt subroutine
#LIMSWI	Label for Limit Switch subroutine
#MCTIME	Label for timeout if encoder is not in-position within time specified by TW.
#POSERR	Label for excess Position Error subroutine
#TCPERR	Ethernet error

Example Program:

#AUTO	Beginning of the Program
SH	Turn motors on
PR 10000,20000;BG XY	Specify relative distances on X and Y axes; Begin Motion
AM	Wait for motion complete
WT 2000	Wait 2 sec
JP # AUTO	Jump to label AUTO
EN	End of Program

The above program will execute automatically at power up and move X and Y 10000 and 20000 units. After the motion is complete, the motors rest for 2 seconds. The cycle repeats indefinitely until the stop command is issued. Automatic execution assumes that the program has been burned in using the BP command.

Executing Programs - Multitasking

Two programs can run independently. The programs (threads) are numbered 0 through 7. 0 is the main thread. The main thread differs from the others in the following points:

1. Only the main thread may use the input command, IN.
2. In a case of interrupts, due to inputs, limit switches, position errors or command errors, it is thread 0 which jumps to those subroutines.

The execution of the various programs is done with the instruction:

XQ #A, n

Where n indicates the thread number. To halt the execution of any thread, use the instruction

HX n

where n is the thread number.

Note that both the XQ (Execute Program) and HX (Halt Execution) functions can be performed by an executing program.

Multitasking is useful for executing independent operations such as PLC functions that occur independently of motion. The example below produces a waveform on Output 1 independent of a move.

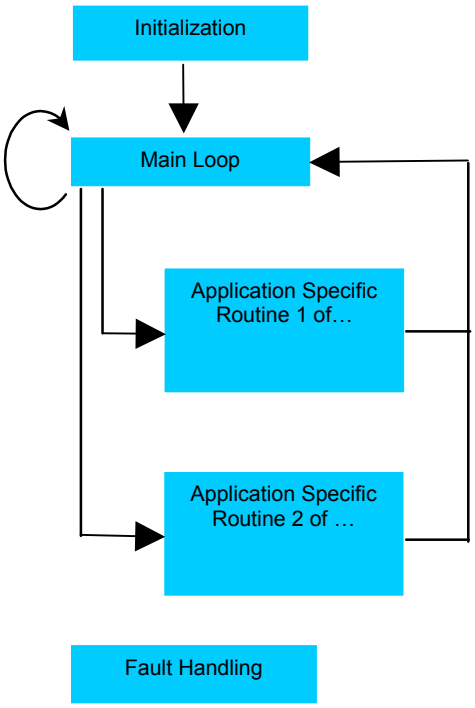
#TASK1	Task1 label
AT0	Initialize reference time
CB1	Clear Output 1
#LOOP1	Loop1 label
AT 10	Wait 10 msec from reference time
SB1	Set Output 1
AT -40	Wait 40 msec from reference time, then initialize reference
CB1	Clear Output 1
JP #LOOP1	Repeat Loop1
#TASK2	Task2 label
XQ #TASK1,1	Execute Task1
#LOOP2	Loop2 label
PR 1000	Define relative distance
BGX	Begin motion
AMX	After motion done
WT 10	Wait 10 msec
JP #LOOP2,@IN[2]=1	Repeat motion unless Input 2 is low
HX	Halt all tasks

The program above is executed with the instruction XQ #TASK2,0 which designates TASK2 as the main thread. #TASK1 is executed within TASK2.

Recommended Programming Style

The SMC controller runs an interpreted, structured text program. The structure is not strictly enforced; the program is a simple text file with a list of instructions. Programs that are not well structured are more difficult to debug and maintain. Yaskawa recommends the following program template layout because it is suitable for any application.

Although most of the examples in the manual are quite short and simple, many applications require much more elaborate programming. Feel free to include blank lines between routines, and use the tab key to indent to make the program more readable. Yterm compresses the blank lines and tab characters out of the program because the controller will not accept them. Remember this, because if a program is uploaded from the controller, it will have lost it's formatting. *Future versions of Yterm will have an auto format function.*



Initialization

Programs should start by initializing any special parameters and user variables that will be used later in the program. Remember that variables must be initialized before use, or a command error will result later in the program. The initialization section contains code that never needs to run after the initial power up. Programmers may choose to incorporate a homing routine near the end of the initialization, just before entering the main loop of the program.

Sometimes programmers will not want some variables to be reinitialized at power up because they contain job specific values. Variables can be saved to flash memory by using the BV command within the program. A handy trick for variables that must retain their values after power cycle is to include their initialization ABOVE the #AUTO label. The variables would only get initialized if the XQ command is issued from the terminal. This is because the XQ command with no argument causes the program to execute at the first line. In contrast, with the #AUTO label is included in the program, program execution starts at the label and continues from there, which would skip those other commands and variables above it.

The following is an example of an initialization section.

```
#AUTO; InitPass=0   //  Auto execute at power up, and init success flag set to false.
//  ------------------  Wizard Shell Code Initialization Section  -------------------
   MOX;  OP0   //  Make sure motor is off and outputs are reset.
   Error=0          //  Initialize Application Wizard error code system
   LimIgnor=0   //  Initialize flag for limit switches during homing.
   TRUE=1;  FALSE=0
   NT=_TM/1024   //   Normalized Time, only required if servo update changed from 1000 uSec default.
// ---------------- Homing Initialization  -----------------
   homing=0        //   1 When homing, 0 when not
   homed=0         //   1 When homed,0 When not homed
   H_SW=-1;  h_sw=-1   //  Active state of the home input  All caps is original config, small is program switchable.
//  ------------------  Rotary Table Initialization   ----------------------------
   DM Accel[9]
   DM ApprDist[9]
   DM ApprOP[9]
   DM Dwell1[9]
   DM Position[9]
   DM PostDWOp[9]
   DM Scurve[9]
   DM Speed[9]
   DM Station[9]
   DM StatOP[9]
   DM Modulus[3], OPosO[3], DL72[3];  i=0       //  For the rotary calculations
   #InitMod; Modulus[i]=0; OPosO[i]=0; DL72[i]=0;  i=i+1; JP #InitMod,i<3
   OldIndex=0; MachCyc=40960.0
   Bit=5; JS #XtoY; BitMaskO=XtoY
   MinBit=5; MaxBit=8; JS #MakeMsk; BitMask=MakeMsk
   WT 500/NT
   SHX; WT500/NT
   InitPass=1      //  Initialization success flag, for special label usage
```

Main Loop

The main loop serves as the control center for all general machine logic. This is typically a small part of the overall program. It is more efficient to include mode-specific logic in lower level routines. This type of logic would include anything that only has any importance while a certain mode is activated. This allows for a faster execution time of the main loop logic, and easier code debugging. The SMC runs an interpreter, which means that the text code of the program is always being translated to machine language command by command. Minimizing the amount of code the controller must execute increases efficiency.

The following is an example of a small main loop:

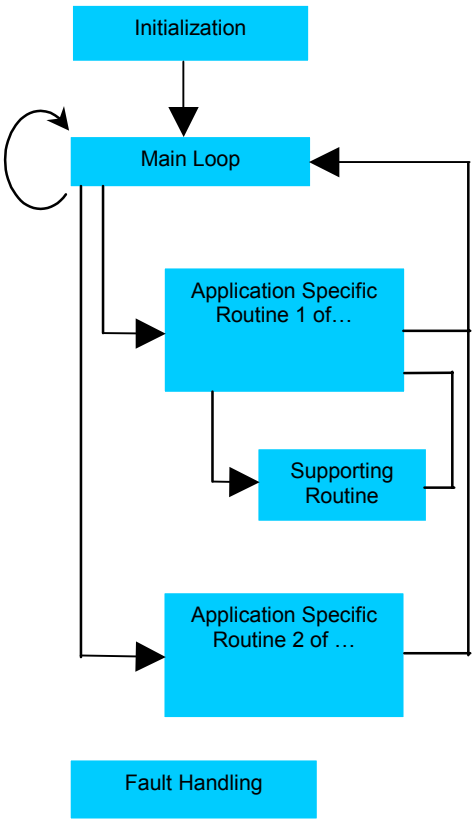
```
#MAIN
   JS #HOME,((@IN[4]=1)&(homed=0))
   Index=(_TI&BitMask)/BitMaskO            // 500-1 determine (by input) what Station to locate
   JS #ROTARY,(Index<=9) & (Index<>OldIndex) & (homed=TRUE)
JP #MAIN
```

Notice that the labels listed here show the modes of operation.

Application Specific Routines

Application specific routines contain the core logic of the machine's modes of operation. The routine only runs once the logic in the main loop has determined that all conditions have been met.

The following is an example of an application specific routine that indexes the servo.



Note: "Automatic Index Subroutine"

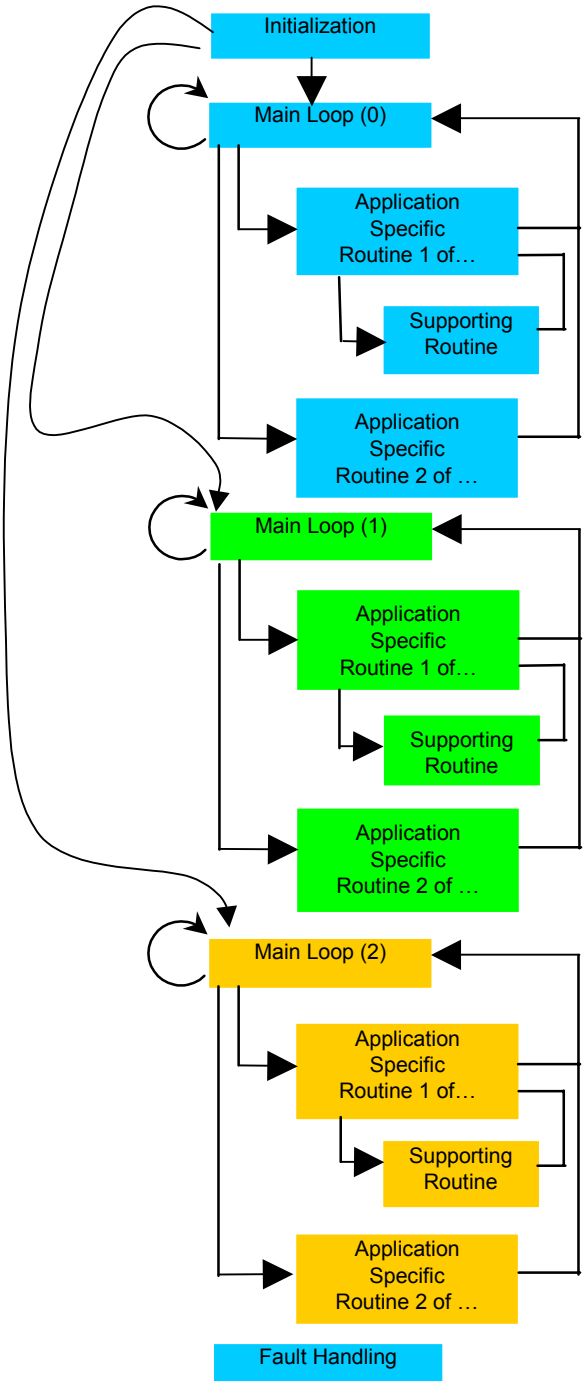
```
#AUTOCYC
    Status=sAuto
    IF (_MOX=1); SHX; WT 1000; ENDIF
#INDEX
    JP #INDEX, ((MODE & IndexGo)=0) & ((MODE & AutoMode)=AutoMode)
    JP #AUTOERR,_RPX<>XHOME
    ACX=IdxDist/2/(IdxTime/3)/(IdxTime/3)+3300; DCX=_ACX
    SPX=IdxDist/2/(IdxTime/3)
    PRX=IdxDist; BGX; Parts=Parts+1; AMX
    SB INDEX1; WT Dwell; CB INDEX1
    #AUTO_DN
EN


#AUTOERR
    Status=sNotHome
    ATHOME=FALSE
    JP #AUTO_DN
EN
```

As shown above, supporting routines or functions of an application specific routine can be included below the routine itself, such as #AUTOERR.

Notice an important point in the #AUTOCYC routine. It is called from the #MAIN loop with a JS (Jump to Subroutine) command. If there was a jump to the #AUTOERR routine, program flow cannot simply return to the #MAIN loop with a JP #MAIN. If this were to happen, the subroutine stack would eventually overflow if this scenario were to happen 16 times. A dummy label #AUTO_DN at the end of the application specific routine allows for a safe return location, so the main routine can end normally and automatically return to the #MAIN loop.

Multi Tasking

Below is a block diagram demonstrating the concept of multitasking. Threads can be started and stopped by each other at will. The only limitation is that the thread being launched with the XQ command cannot be already executing. The recommended method is to start threads in the initialization section of the program. Once started, they will continue to operate until halted. The controller will time share between threads on a line-by-line basis. If one program has more commands per line, it will receive more of the total execution time available.



Multi tasking is a very useful method of accomplishing several tasks that have very little or nothing do with each other. Focus can be specifically on the task, whereas extra "juggling" code would be necessary to adequately handle several unrelated events in a single program loop. Good examples of multitasking include:

- Software limit updates for two axes that operate in each others zone where a collision could occur.
- An analog output must be updated continuously
- Short-term event that must be monitored at the same time as another process. This is a "spur" thread that runs and dies after a certain event is attained, such as a timer.
- I/O logic must be solved at a regular interval.
- Events that may take place at the same time, but are handled at different times, such as a registration buffer that must store product positions, but the correction for a particular product will occur slightly later.

There is no need to multitask when moving two or more servos if these axes always move in a predefined sequence. Some users incorrectly assume that they must use a thread per axis. Experience will dictate when it is better to use multitasking, and when one task will suffice.

Subroutines and other sections of code should not be intermixed between threads to avoid complexity and possible erratic behavior.

All variables & arrays defined in the program are global, meaning that all threads can read and write the data. This is very useful when threads must share information, such as a product registration buffer. This can be troubling if the programmer unknowingly uses the same variable in both threads for something simple, such as the variable "X" as a counter.

Special Label & Fault Handling Routines

Fault Handling routines are the special labels that automatically execute without being referred to in the program with a JP or JS. They are the following

#AUTO	Auto program execute at power up or reset.
#ININT	Input interrupt for any combination of local inputs
#CMDERR	Command error in program handler
#POSERR	Excessive position error
#LIMSWI	Limit switch
#TCPERR	TCP Error
#MCTIME	Motion complete timeout

The #CMDERR, #POSERR, and #LIMSWI routines are highly recommended for all programs to greatly increase their robustness. It is very important to integrate Special Label routines with care by thinking about the program logic required to make the machine run properly. The following are some DO's and DON'Ts:

DO	DON'T
Abort motion for safety if a #CMDERR occurs. Use AB1	Don't return to the program using RE if a #CMDERR occurs.
Under #CMDERR, include a method for the error code (_TC) and line number (_ED) to be identified by someone who can help debug the program. Store them in a variable and present them on an operator screen.	Don't call subroutines from within the #ININT special routine and forget to do an RI. This would leave the controller thinking it is handling the interrupt, and additional interrupts cannot occur. Keep interrupts simple and short.
Check for software and hardware limits for each axis in the #LIMSWI routine.	Don't set the software limits FL and BL to the same value; the servo will not be able to move once it crosses that position.
	Don't use the ZS command to correct programming problems with the JS and JP commands.

Other considerations

- There is a chance that the special label routines could execute just after power up, but before the initialization, section has completed. If any of the special routines jump back to the #MAIN loop after recovery, problems could occur because of items that were "skipped" in the initialization. To avoid this, use a flag such as the one in the above examples called "InitPass" and check it's value when deciding where to resume after a special label event. It is possible that the servo could have following error when the program starts, and the program will immediately jump to #POSERR. Once the following error is cleared, the #POSERR must check the initialization flag and jump back to the top of the program to attempt to initialize again.
- If a Command Error occurred in the #CMDERR special label routine, it will recursively call itself (to indicate the additional command errors), and eventually cause a stack overflow. This condition may cause the controller to behave strangely. Be extra cautious when programming the #CMDERR routine. Minimize the code in the routine to limit the chance of creating a programming error. If you suspect a #CMDERR in the #CMDERR routine, place an MG command as the first command within the routine so you can see the message print out at the terminal screen of Yterm each time an error occurs.
- If a thread other than thread zero causes a command error, this can be determined in code by using the _ED1 parameters. If another thread had an error, it may be possible to restart the thread although many errors will eventually require a program fix. Use the HX and XQ commands within the #CMDERR routine to restart the thread at it's main label. Usually it is too complex to have a thread resume after a fault. Remember that all special label routines run as subroutines of thread zero, so in this case, an RE command will be necessary for thread zero to resume.

Refer to section 10 – "Example Applications" at the back of the manual for a detailed discussion of the Special Label error routines.

Debugging Programs

The SMC–4000 provides trace and error code commands which are used for debugging programs. The trace command may be activated using the command, TR1. This command causes each line in a program to be sent out to the communications port immediately prior to execution. The TR1 command is useful for debugging programs. TR0 disables the trace function. The TR command may also be included as part of a program.

If there is a program error, the SMC–4000 will halt program execution at the line number at which an error occurs and display the line. The user can obtain information about the type of error condition that occurred by using the command, TC1. Check the TC (Tell Code) command reference page for a complete listing of the codes.

Program Flow Commands

The SMC–4000 provides instructions that control program flow. The SMC–4000 program sequencer executes instructions in a program sequentially. Program Flow commands, however, may be used to redirect program flow. A summary of these commands is given below and they are detailed in the following sections.

Program Flow Command Summary

AD	After Distance Trigger
AI	After Input Trigger
AM	After Motion Complete Trigger
AP	After Absolute Position Trigger
AR	Relative Distance Trigger
AS	After Speed Trigger
AT	Wait for time with respect to reference
AV	After Vector Distance Trigger
ELSE	ELSE Function for use with IF Conditional Statement
ENDIF	End of IF Conditional Statement
IF	IF Conditional Statement
JP	Conditional Jump
JS	Conditional Jump to Subroutine
MC	Trigger "In position" trigger (TW x,y,z,w sets timeout for in-position)
MF	Trigger Forward motion
MR	Trigger Reverse motion
WC	Wait for Contour Data
WT	Wait for time to elapse

Event Triggers & Trippoints

To function independently from the host computer, the SMC–4000 can be programmed to make decisions based on the occurrence of an event. Such events include waiting for motion to be complete, waiting for a specified amount of time to elapse, or waiting for an input to change logic levels.

The SMC–4000 provides several event triggers that cause the program sequencer to halt until the specified event occurs. Normally, a program is automatically executed sequentially one line at a time. When an event trigger instruction is decoded, however, the actual program sequence is halted. The program sequence does not continue until the event trigger is "tripped". For example, the motion complete trigger can be used to separate two move sequences in a program. The commands for the second move sequence will not be executed until the motion is complete on the first motion sequence. In this way, the SMC–4000 can make decisions based on its own status or external events without intervention from a host computer.

SMC–4000 Event Triggers

Command	Function
AM X Y Z W or S (A B C D E F G H)	Halts program execution until motion is complete on the specified axes or motion sequence(s). AM with no parameter tests for motion complete on all axes. This command is for separating motion sequences in a program.
AD X or Y or Z or W (A or B or C or D or E or F or G or H)	Halts program execution until position command has reached the specified relative distance from the start of the move. Only one axis may be specified at a time.
AR X or Y or Z or W (A or B or C or D or E or F or G or H)	Halts program execution until after specified distance from the last AR or AD command has elapsed. Only one axis may be specified at a time.
AP X or Y or Z or W (A or B or C or D or E or F or G or H)	Halts program execution until after absolute position occurs. Only one axis may be specified at a time.
AI +/-n	Halts program execution until after specified input is at specified logic level. n specifies input line. Positive is high logic level, negative is low level.
AS X Y Z W S (A B C D E F G H)	Halts program execution until specified axis has reached its slew speed.
AT +/-n	Halts program execution until n msec from reference time. AT 0 sets reference. AT n waits n msec from reference. AT -n waits n msec from reference and sets new reference after elapsed time.
AV n	Halts program execution until specified distance along a coordinated path has occurred.
MC X or Y or Z or W (A or B or C or D or E or F or G or H)	Halt program execution until after the motion profile has been completed and the encoder has entered or passed the specified position. TW x,y,z,w sets timeout to declare an error if not in position. If timeout occurs, then the trippoint will clear and the stop code will be set to 99. An application program will jump to label #MCTIME.
MF X or Y or Z or W (A or B or C or D or E or F or G or H)	Halt program execution until after forward motion reached absolute position. Only one axis may be specified. If position is already past the point, then MF will trip immediately.

Command	Function
MR X or Y or Z or W (A or B or C or D or E or F or G or H)	Halt program execution until after reverse motion reached absolute position. Only one axis may be specified. If position is already past the point, then MR will trip immediately.
WT n	Halts program execution until specified time in msec has elapsed.

Event Trigger Examples:

Event Trigger - Multiple Move Sequence

The AM trippoint is used to separate the two PR moves. If AM is not used, the controller returns a ? for the second PR command because a new PR cannot be given until motion is complete.

#TWOMOVE	Label
PR 2000	Position Command
BGX	Begin Motion
AMX	Wait for Motion Complete
PR 4000	Next Position Move
BGX	Begin 2nd move
EN	End program

In the above example, the AM trippoint is used to separate the two PR moves. If AM is not used, the controller returns a ? for the second PR command because a new PR cannot be given until motion is complete.

Event Trigger - Set Output after Distance

Set output bit 1 after a distance of 1000 counts from the start of the move. The accuracy of the trippoint is the speed multiplied by the sample period.

#SETBIT	Label
SP 10000	Speed is 10000
PA 20000	Specify Absolute position
BGX	Begin motion
AD 1000	Wait until 1000 counts
SB1	Set output bit 1
EN	End program

The above example sets output bit 1 after a distance of 1000 counts from the start of the move. The accuracy of the trippoint is the speed multiplied by the sample period.

Event Trigger - Repetitive Position Trigger

To set the output bit every 10000 counts during a move, the AR trippoint is used shown in the next example.

#TRIP	Label
JG 50000	Specify Jog Speed
BGX;N=0	Begin Motion
#REPEAT	# Repeat Loop
AR 10000	Wait 10000 counts
TPX	Tell Position
SB1	Set output 1
WT50	Wait 50 msec
CB1	Clear output 1
N=N+1	Increment counter
JP #REPEAT,N<5	Repeat 5 times
STX	Stop
EN	End

Event Trigger - Start Motion on Input

This example waits for input 1 to go low and then starts motion. Note: The AI command actually halts execution of the program until the input occurs. If you do not want to halt the program sequences, you can use the Input Interrupt function (II) or use a conditional jump on an input, such as JP #GO,@IN[1] = 0.

#INPUT	Program Label
AI-1	Wait for input 1 low
PR 10000	Position command
BGX	Begin motion
EN	End program

Event Trigger - Set Output when at Speed

#ATSPEED	Program Label
JG 50000	Specify jog speed
AC 10000	Acceleration rate
BGX	Begin motion
ASX	Wait for at slew speed 50000
SB1	Set output 1
EN	End program

Event Trigger - Change Speed along Vector Path

The following program changes the feed rate or vector speed at the specified distance along the vector. The vector distance is measured from the start of the move or from the last AV command.

#VECTOR	Label
VMXY;VS 5000	Coordinated path
VP 10000,20000	Vector position
VP 20000,30000	Vector position
VE	End vector
BGS	Begin sequence
AV 5000	After vector distance
VS 1000	Reduce speed
EN	End

Event Trigger - Multiple Move with Wait

#MOVES	Label
PR 12000	Distance
SP 20000	Speed
AC 100000	Acceleration
BGX	Start Motion
AD 10000	Wait a distance of 10,000 counts
SP 5000	New Speed
AMX	Wait until motion is completed
WT 200	Wait 200 ms
PR -10000	New Position
SP 30000	New Speed
AC 150000	New Acceleration
BGX	Start Motion
EN	End

Define Output Waveform Using AT

The following program causes Output 1 to be high for 10 msec and low for 40 msec. The cycle repeats every 50 msec.

#OUTPUT	Program label
AT0	Initialize time reference
SB1	Set Output 1
#LOOP	Loop
AT 10	After 10 msec from reference,
CB1	Clear Output 1
AT -40	Wait 40 msec from reference and reset reference
SB1	Set Output 1
JP #LOOP	Loop
EN	

Conditional Jumps

The SMC–4000 provides Conditional Jump (JP) and Conditional Jump to Subroutine (JS) instructions for branching to a new program location based on a specified condition. Unlike event triggers, the conditional jump instruction does not halt the program sequence. Instead, it tests to see if a condition is satisfied and then branches to a new location or subroutine. (A subroutine is a group of commands defined by a label and EN command. After all the commands in the subroutine are executed, a return is made to the main program). If the condition is not satisfied, the program sequence continues to the next program line.

The JP and JS instructions have the following format:

Format:	Meaning
JS destination, logical condition	Jump to subroutine if logical condition is satisfied
JP destination, logical condition	Jump to location if logical condition is satisfied

The destination is a program line number or label. The destination is where the program sequencer jumps to if the specified condition is satisfied. The comma designates "IF". The logical condition tests two operands with logical operators. The operands can be any valid SMC–4000 numeric operand, including variables, array elements, numeric values, functions, keywords, and arithmetic expressions.

Logical operators:

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

Operands:

Type	Examples
Number	V1=6
Numeric Expression	V1=V7*6
	@ABS[V1]>10
Array Element	V1<Count[2]
Variable	V1<V2
Internal Variable	_TPX=0
	_TVX>500
I/O	V1>@AN[2]
	@IN[1]=0

The jump statement may also be used without a condition.

Example of conditional jump statements are given below:

Conditional	Meaning
JP #LOOP,COUNT<10	Jump to #LOOP if the variable, COUNT, is less than 10
JS #MOVE2,@IN[1]=1	Jump to subroutine #MOVE2 if input 1 is logic level high. After the subroutine MOVE2 is executed, the program sequencer returns to the main program location where the subroutine was called.
JP #BLUE,@ABS[V2]>2	Jump to #BLUE if the absolute value of variable, V2, is greater than 2
JP #C,V1*V7<=V8*V2	Jump to #C if the value of V1 times V7 is less than or equal to the value of V8*V2
JP#A	Jump to #A

Conditional jumps are useful for testing events in real-time. They allow the SMC–4000 to make decisions without a host computer. For example, the SMC–4000 can decide between two motion profiles based on the state of an input line. Or, the SMC–4000 can keep track of how many times a motion profile is executed.

Example:

Move the X motor to absolute position 1000 counts and back to zero ten times. Wait 100 msec between moves.

#BEGIN	Begin Program
COUNT=10	Initialize loop counter
#LOOP	Begin loop
PA 1000	Position absolute 1000
BGX	Begin move
AMX	Wait for motion complete
WT 100	Wait 100 msec
PA 0	Position absolute 0
BGX	Begin move
AMX	Wait for motion complete
WT 100	Wait 100 msec
COUNT=COUNT-1	Decrement loop counter
JP #LOOP,COUNT>0	Test for 10 times through loop
EN	End Program

Multiple Conditional Statements

The SMC–4000 will accept multiple conditions in a single jump statement. The conditional statements are combined in pairs using the operands "&" and "|". The "&" operand between any two conditions, requires that both statements must be true for the combined statement to be true. The "|" operand between any two conditions, requires that only one statement be true for the combined statement to be true.

***Note Each condition must be placed in parentheses for proper evaluation by the controller. In addition, the SMC–4000 executes operations from left to right.**

Example using variables:

JP #TEST, (V1<V2) & (V3<V4)

In this example, this statement will cause the program to jump to the label #TEST if V1 is less than V2 and V3 is less than V4. To illustrate this further, consider this same example with an additional condition:

JP #TEST, ((V1<V2) & (V3<V4)) | (V5<V6)

This statement will cause the program to jump to the label #TEST under two conditions; 1. If V1 is less than V2 and V3 is less than V4. OR 2. If V5 is less than V6.

Examples

If the condition for the JP command is satisfied, the controller branches to the specified label or line number and continues executing commands from this point. If the condition is not satisfied, the controller continues to execute the next commands in sequence.

Format	Meaning
JP #Loop, COUNT<10	Jump to #Loop if the variable, COUNT, is less than 10
JS #MOVE2,@IN[1]=1	Jump to subroutine #MOVE2 if input 1 is logic level high. After thesubroutine MOVE2 is executed, the program sequencer returns to the main program location where the subroutine was called.
JP #BLUE,@ABS[V2]>2	Jump to #BLUE if the absolute value of variable, V2, is greater than 2
JP #C,V1*V7<=V8*V2	Jump to #C if the value of V1 times V7 is less than or equal to the value of V8*V2
JP#A	Jump to #A

Example:

Move the A motor to absolute position 1000 counts and back to zero ten times. Wait 100 msec between moves.

#BEGIN	Begin Program
COUNT=10	Initialize loop counter
#LOOP	Begin loop
PA 1000	Position absolute 1000
BGA	Begin move
AMA	Wait for motion complete
WT 100	Wait 100 msec
PA 0	Position absolute 0
BGA	Begin move
AMA	Wait for motion complete
WT 100	Wait 100 msec
COUNT=COUNT-1	Decrement loop counter
JP #LOOP,COUNT>0	Test for 10 times thru loop
EN	End Program

If, Else, and Endif

The SMC–4000 provides a structured approach to conditional statements using IF, ELSE and ENDIF commands.

Using the IF and ENDIF Commands

An IF conditional statement is formed by the combination of an IF and ENDIF command. The IF command has as it's arguments one or more conditional statements. If the conditional statement(s) evaluates true, the command interpreter will continue executing commands which follow the IF command. If the conditional statement evaluates false, the controller will ignore commands until the associated ENDIF command is executed OR an ELSE command occurs in the program (see discussion of ELSE command below).

NOTE: An ENDIF command must always be executed for every IF command that has been executed. It is recommended that the user not include jump commands inside IF conditional statements since this causes re-direction of command execution. In this case, the command interpreter may not execute an ENDIF command.

NOTE: Do not jump (JP) out of an IF block. If this occurs, the ENDIF instruction will never be executed.

Using the ELSE Command

The ELSE command is an optional part of an IF conditional statement and allows for the execution of command only when the argument of the IF command evaluates False. The ELSE command must occur after an IF command and has no arguments. If the argument of the IF command evaluates false, the controller will skip commands until the ELSE command. If the argument for the IF command evaluates true, the controller will execute the commands between the IF and ELSE command.

Nesting IF Conditional Statements

The SMC–4000 allows IF conditional statements to be included within other IF conditional statements. This technique is known as 'nesting' and the SMC–4000 allows up to 255 IF conditional statements to be nested. This is a very powerful technique allowing the user to specify a variety of different cases for branching.

Command Format - IF, ELSE and ENDIF

Format:	Meaning
IF <condition>	Execute commands proceeding IF command (up to ELSE command) if conditional statement(s) is true, otherwise continue executing at ENDIF command or optional ELSE command.
ELSE	Optional command. Allows for commands to be executed when argument of IF command evaluates not true. Can only be used with IF command.
ENDIF	Command to end IF conditional statement. Program must have an ENDIF command for every IF command.

Example:

#TEST	Begin Main Program "TEST"
II,,3	Enable interrupts on input 1 and input 2
MG "WAITING FOR INPUT 1, INPUT 2"	Output message
#LOOP	Label to be used for endless loop
JP #LOOP	Endless loop
EN	End of main program
#ININT	Input Interrupt Subroutine
IF (@IN[1]=0)	IF conditional statement based on input 1
IF (@IN[2]=0)	2nd IF executed if 1st IF conditional true
MG "INPUT 1 AND INPUT 2 ARE ACTIVE"	Message executed if 2nd IF is true
ELSE	ELSE command for 2nd IF statement
MG "ONLY INPUT 1 IS ACTIVE	Message executed if 2nd IF is false
ENDIF	End of 2nd conditional statement
ELSE	ELSE command for 1st IF statement
MG"ONLY INPUT 2 IS ACTIVE"	Message executed if 1st IF statement
ENDIF	End of 1st conditional statement
#WAIT	Label to be used for a loop
JP#WAIT,(@IN[1]=0) | (@IN[2]=0)	Loop until Input 1& 2 are not active
RI0	End Input Interrupt Routine without restoring trippoints

Subroutines

A subroutine is a group of instructions beginning with a label and ending with an END (EN). Subroutines are called from the main program with the jump subroutine instruction JS, followed by a label or line number, and conditional statement. Up to 16 subroutines can be nested. After the subroutine is executed, the program sequencer returns to the program location where the subroutine was called unless the subroutine stack is manipulated as described in the following section.

Example:

An example of a subroutine to draw a square 500 counts per side is given below. The square is drawn at vector position 1000,1000.

#M	Begin Main Program
CB1	Clear Output Bit 1 (pick up pen)
VP 1000,1000;LE;BGS;AMS	Define vector position; move pen
SB1	Set Output Bit 1 (put down pen)
JS #SQUARE;CB1	Jump to square subroutine
EN	End Main Program
#SQUARE	Square subroutine
V1=500;JS #L	Define length of side
V1=-V1;JS #L	Switch direction
EN	End subroutine
#L;PR V1,V1;BGX	Define X,Y; Begin X
AMX;BGY;AMY	After motion on X, Begin Y
EN	End subroutine

Stack Manipulation

It is possible to manipulate the subroutine stack by using the ZS command. Every time a JS instruction, interrupt or automatic routine (such as #POSERR or #LIMSWI) is executed, the subroutine stack is incremented by 1. Normally the stack is restored with an EN instruction. Occasionally it is desirable not to return back to the program line where the subroutine or interrupt was called. The ZS1 command clears 1 level of the stack. This allows the program sequencer to continue to the next line. The ZS0 command resets the stack to its initial value. For example, if a limit occurs and the #LIMSWI routine is executed, it is often desirable to restart the program sequence instead of returning to the location where the limit occurred. To do this, give a ZS command at the end of the #LIMSWI routine.

Auto Start Routine

#AUTO	Auto start program on power-up

If the #AUTO label is included in a Burned Program (BP command), the controller will start executing the program starting at the location of the #AUTO label when power is applied.

Automatic Subroutines for Monitoring Conditions

Often it is desirable to monitor certain conditions continuously without tying up the host or SMC–4000 program sequences. The SMC–4000 can monitor several important conditions in the background. These conditions include checking for the occurrence of a limit switch, a defined input, position error, or a command error. Automatic monitoring is enabled by inserting a special, pre-defined label in the applications program. The pre-defined labels are:

#CMDERR	Bad command given
#ININT	Input specified by II goes low
#LIMSWI	Limit switch on any axis goes low
#MCTIME	Timeout for In-position trippoint, MC
#POSERR	Position error exceeds limit specified by ER
#TCPERR	Ethernet error

For example, the #POSERR subroutine will automatically be executed when any axis exceeds its position error limit. The commands in the #POSERR subroutine could decode which axis is in error and take the appropriate action. In another example, the #ININT label could be used to designate an input interrupt subroutine. When the specified input occurs, the program will be executed automatically.

***Note An application program must be running for automatic monitoring to function.**

Example - Limit Switch:

This program prints a message upon the occurrence of a limit switch. Note, for the #LIMSWI routine to function, the SMC–4000 must be executing an applications program from memory. This can be a very simple program that does nothing but loop on a statement, such as #LOOP;JP #LOOP;EN. Motion commands, such as JG 5000 can still be sent from the PC even while the "dummy" applications program is being executed.

#LOOP	Dummy Program
JP #LOOP;EN	Jump to Loop
#LIMSWI	Limit Switch Label
MG "LIMIT OCCURRED"	Print Message
RE	Return to main program
:XQ #LOOP	Execute Dummy Program
:JG 5000	Jog
:BGX	Begin Motion

Now, when a forward limit switch occurs on the X axis, the #LIMSWI subroutine will be executed.

***Note The RE command is used to return from the #LIMSWI subroutine.**

***Note The #LIMSWI will continue to be executed until the limit switch is cleared (goes high).**

Example - Position Error

#LOOP	Dummy Program

JP #LOOP;EN	Loop
#POSERR	Position Error Routine
V1=_TEX	Read Position Error
MG "EXCESS POSITION ERROR"	Print Message
MG "ERROR=",V1=	Print Error
RE	Return from Error
:XQ #LOOP	Execute Dummy Program
:JG 100000	Jog at High Speed
:BGX	Begin Motion

If the position error on the X axis exceeds that specified by the ER command, the #POSERR routine will execute.

***Note The RE command is used to return from the #POSERR subroutine**

***Note The #POSERR routine will continue to be executed until the position error is cleared (is less than the ER limit).**

Input Interrupt Example:

#A	Label
II1	Input Interrupt on 1
JG 30000,,,60000	Jog
BGXW	Begin Motion
#LOOP;JP#LOOP;EN	Loop
#ININT	Input Interrupt
ST;AM	Stop Motion
#TEST;JP #TEST, @IN[1]=0	Test for Input 1 still low
BGXW;RI	Begin motion and Return to Main Program
EN	

***Note Use the RI command to return from #ININT subroutine.**

Bad Command Example

#BEGIN	Begin main program
IN "ENTER SPEED", SPEED	Prompt for speed
JG SPEED;BGX;	Begin motion
JP #BEGIN	Repeat
EN	End main program
#CMDERR	Command error utility
JP#DONE,_TC<>6	Check if out of range
MG "SPEED TOO HIGH"	Send message
MG "TRY AGAIN"	Send message
ZS1	Adjust stack
JP #BEGIN	Return to main program
#DONE	End program if other error
ZS0	Zero stack
EN	End program

The above program prompts the operator to enter a jog speed. If a number is entered out of range (greater than 12 million), the #CMDERR routine will be executed prompting the operator to enter a new number.

Mathematical and Functional Expressions

For manipulation of data, the SMC–4000 provides the use of the following mathematical operators:

Operator	Function
+	Addition
-	Subtraction
*	Multiplication
/	Division
&	Logical And (Bit-wise)
|	Logical Or (On some computers, a solid vertical line appears as a broken line)
( )	Parenthesis

The numeric range for addition, subtraction and multiplication operations is +/-2,147,483,647.9999. The precision for division is 1/65,000.

Mathematical operations are executed from left to right. Parentheses can be used and nested four deep. Calculations within a parentheses have precedence.

Examples:

SPEED=7.5*V1/2	The variable, SPEED, is equal to 7.5 multiplied by V1 and divided by 2
COUNT=COUNT+2	The variable, COUNT, is equal to the current value plus 2.
RESULT=_TPX-(@COS[45]*40)	Puts the position of X - 28.28 in RESULT. 40 * cosine of 45° is 28.28
TEMP=@IN[1]&@IN[2]	TEMP is equal to 1 only if Input 1 and Input 2 are high

The SMC–4000 also provides the following functions:

Function	Command Meaning
@ABS[n]	Absolute Value
@ACOS[n]	Arc Cosine
@AN[n]	Read analog input n
@ASIN[n]	Arc Sine
@ATAN[n]	Arc Tangent
@COM[n]	2's Complement
@COS[n]	Cosine
@FRAC[n]	Fraction
@IN[n]	Read digital input n
@INT[n]	Integer
@OUT[n]	Output state
@RND[n]	Rounds number .5 and up to next integer
@SIN[n]	Sine
@SQR[n]	Square Root Function; Accuracy is +/-.0004
@TAN[n]	Tangent

Functions may be combined with mathematical expressions. The order of execution is from left to right. The units of the SIN and COS functions are in degrees with resolution of 1/128 degrees. The values can be up to +/-2 billion degrees.

Example:

V1=@ABS[V7]	The variable, V1, is equal to the absolute value of variable V7.
V2=5*@SIN[POS]	The variable, V2, is equal to five times the sine of the variable, POS.
V3=@IN[1]	The variable, V3, is equal to the digital value of input 1.
V4=@AN[5]	The variable, V4, is equal to the digital value of analog input 5.

Variables

Many motion applications include parameters that are variable. For example, a cut-to-length application often requires that the cut length be variable. The motion process is the same, however the length is changing.

To accommodate these applications, the SMC–4000 provides for the use of both numeric and string variables. A program can be written in which certain parameters, such as position or speed, are defined as variables. The variables can later be assigned by the operator or determined by the program calculations.

All variables created in the SMC are 48 bit fixed decimal point data. 32 bits are integer (+/- 2147483647) and 16 bits are fraction (1/65535)

Example:

PR POSX	Assigns variable POSX to PR command
JG RPMY*70	Assigns variable RPMY multiplied by 70 to JG command.

Programmable Variables

The SMC–4000 allows the user to create up to 510 variables. Each variable is defined by a name which can be up to eight characters. The name must start with an alphabetic character, however, numbers are permitted in the rest of the name. Spaces are not permitted. Examples of valid and invalid variable names are:

Valid Variable Names

POSX

POS1

SPEEDZ

Invalid Variable Names

1POS

123

SPEED Z

It is recommended that variable names not be the same as SMC–4000 instructions. For example, PR is not a good choice for a variable name.

The range for numeric variable values is 4 bytes of integer followed by two bytes of fraction (+/-2,147,483,647.9999).

String variables can contain up to six characters which must be in quotation. Example: VAR="STRING".

Numeric values can be assigned to programmable variables using the equal sign. Assigned values can be numbers, internal variables and keywords, and functions. String values can be assigned to variables using quotations.

Any valid SMC–4000 function can be used to return a value such as V1=@ABS[V2] or V2=@IN[1]. Arithmetic operations are also permitted.

Example:

POSX=_TPX	Assigns returned value from TPX command to variable POSX.
SPEED=5.75	Assigns value 5.75 to variable SPEED
INPUT=@IN[2]	Assigns logical value of input 2 to variable INPUT
V2=V1+V3*V4	Assigns the value of V1 plus V3 times V4 to the variable V2.
VAR="CAT"	Assign the string, CAT, to VAR

Variable values may be assigned to controller parameters such as GN or PR. Here, an equal is not used. For example:

PR V1	Assign V1 to PR command
SP VS*2000	Assign VS*2000 to SP command

Example - Using Variables for Joystick

The example below reads the voltage of an X-Y joystick and assigns it to variables VX and VY to drive the motors at proportional velocities, where

10 Volts = 8191 counts --> 3000 rpm = 200000 c/sec

Speed/Analog input = 200000/8191 = 24.4

#JOYSTICK	Label
JG 0,0	Set in Jog mode
BGXY	Begin Motion
#LOOP	Loop
VX=@AN[1]*24.4	Read joystick X
VY=@AN[2]*24.4	Read joystick Y
JG VX,VY	Jog at variable VX,VY
JP#LOOP	Repeat
EN	End

Internal Variables & Keywords

Internal variables allow motion or status parameters from SMC–4000 commands to be incorporated into programmable variables and expressions. Internal variables are designated by adding an underscore (_) prior to the SMC–4000 command. SMC–4000 commands which can be used as internal variables are listed in the Command Reference as "Used as an Operand".

Most SMC–4000 commands can be used as internal variables. Status commands such as Tell Position return actual values, whereas action commands such as GN or SP return the values in the SMC–4000 registers. The X,Y,Z or W or A,B,C,D,E,F,G,H for the SMC–4000, axis designation is required following the command.

Examples:

POSX=_TPX	Assigns value from Tell Position X to the variable POSX.
JP #LOOP,_TEX>5	Jump to #LOOP if the position error of X is greater than 5
JP #ERROR,_TC=1	Jump to #ERROR if the error code equals 1.

Internal variables can be used in an expression and assigned to a programmable variable, but they cannot be assigned a value. For example: _KDX=2 is invalid.

The SMC–4000 also provides a few keywords which give access to internal variables that are not accessible by standard SMC–4000 commands.

Keyword	Function
_BGX or _BGY or _BGW	Motion Done if 1. Moving if 0.
_LFX or _LFY or _LFZ or_LFW	Forward Limit (equals 0 or 1)
_LRX or _LRY or _LRZ or LRW	Reverse Limit (equals 0 or 1)
TIME	Free-Running Real Time Clock* (off by 2.4% - Reset on power-on). Note: TIME does not use _.
_HMX or _HMY or _HMZ or HMW	Home Switch (equals 0 or 1)

Examples:

V1=_LFX	Assign V1 the logical state of the Forward Limit Switch on the X-axis
V3=TIME	Assign V3 the current value of the time clock
V4=_HMW	Assign V4 the logical state of the Home input on the W-axis

Example Program:

#TIMER	Timer
INITIME=TIME	Initialize time variable
PR50000;BGX	Begin move
AMX	After move
ELAPSED=TIME-INTIME	Compute elapsed time
EN	End program
#LIMSWI	Limit Switch Routine
JP #FORWARD,_LFX=0	Jump if Forward Limit
AMX	Wait for Motion Done
PR 1000;BGX;AMX	Move Away from Reverse Limit
JP #END	Exit
#FORWARD	Forward Label
PR -1000;BGX;AMX	Move Away from Forward Limit
#END	Exit
RE	Return to Main Program

Arrays

For storing and collecting numerical data, the SMC–4000 provides array space for 15000 elements in up to 30 arrays. Arrays can be used to capture real-time data, such as position, torque and analog input values. In the contouring mode, arrays are convenient for learning a position trajectory and later playing it back.

Defining Arrays

An array is defined by a name and number of entries using the DM command. The name can contain up to eight characters, starting with an uppercase alphabetic character.

The number of entries in the defined array is enclosed in [ ].

Up to 30 different arrays may be defined. The arrays are one dimensional.

All array elements have the same structure as variables, 48 bit fixed decimal point.

Example:

DM POSX[7]	Defines an array named POSX with seven entries
DM SPEED[100]	Defines an array named speed with 100 entries
DM POSX[0]	Frees array space

Each array element has a numeric range of 4 bytes of integer (2^{31})followed by two bytes of fraction (+/-2,147,483,647.9999).

Array space may be de-allocated using the DA command followed by the array name. DA*[0] de-allocates all the arrays.

Assignment of Array Entries

Like variables, each array element can be assigned a value. Assigned values can be numbers or returned values from instructions, functions and keywords.

Values are assigned to array entries using the equal sign. Assignments are made one element at a time by specifying the element number with the associated array name.

***Note Remember to define arrays using the DM command before assigning entry values.**

Example:

DM SPEED[10]	Dimension Speed Array
SPEED[1]=7650.2	Assigns the first element of the array, SPEED the value 7650.2
SPEED[1]=	Returns array element value
POSX[10]=_TPX	Assigns the 10th element of the array POSX the returned value from the tell position command.
CON[2]=@COS[POS]*2	Assigns the second element of the array CON the cosine of the variable POS multiplied by 2.
TIMER[1]=TIME	Assigns the first element of the array timer the returned value of the TIME keyword.

An array element number can also be a variable. This allows array entries to be assigned sequentially using a counter.

Example:

#A	Begin Program
COUNT=0;DM POS[10]	Initialize counter and define array
#LOOP	Begin loop
WT 10	Wait 10 msec
POS[COUNT]=_TPX	Record position into array element
POS[COUNT]=	Report position
COUNT=COUNT+1	Increment counter
JP #LOOP,COUNT<10	Loop until 10 elements have been stored
EN	End Program

The above example records 10 position values at a rate of one value per 10 msec. The values are stored in an array named POS. The variable, COUNT, is used to increment the array element counter. The above example can also be executed with the automatic data capture feature described as follows.

Arrays may be uploaded and downloaded using the QU and QD commands.

QU array[ ],start,end,comma

QD array[ ],start,end

where array is an array name such as A[ ].

Start is the first element of array (default=0)

End is the last element of array (default=last element)

Comma -- if comma is a 1, then the array elements are separated by a comma. If not a 1, then the elements are separated by a carriage return.

The file is terminated using <control>Z, <control>Q, <control>D or \.

Automatic Data Capture into Arrays

The SMC–4000 provides a special feature for automatic capture of data such as position, position error, inputs or torque. This is useful for teaching motion trajectories or observing system performance. Up to eight types of data can be captured and stored in eight arrays. The capture rate or time interval may be specified.

Commands used:

RA n[ ],m[ ],o[ ],p[ ]	Selects up to four arrays for data capture. The arrays must be defined with the DM command.
RD_TI,_TPX,_SCZ,_TSY	Selects the type of data to be recorded. See the table below for the various types of data. The order of data type is important and corresponds with the order of n,m,o,p arrays in the RA command. In this example, the _TI input data is stored in the first array selected by the RA command.
RC n,m	The RC command begins data collection. Sets data capture time interval where n is an integer between 1 and 8 and designates 2n msec between data. m is optional and specifies the number of elements to be captured. If m is not defined, the number of elements defaults to the smallest array defined by DM. n=0 stops recording.
RC? or V=_RC	Returns a 0 or 1 where, 0 denotes not recording, 1 specifies recording in progress

Data Types for Recording

_DEX	2nd encoder position (dual encoder)
_TPX	Encoder position
_TEX	Position error
_RPX	Commanded position
_RLX	Latched position
_TI	Inputs
_OP	Output
_TSX	Switches (only bit 0-4 valid)
_SCX	Stop code
_TBX	Status bits
_TTX	Torque (reports digital value +/-32703)

***Note X may be replaced by Y,Z or W for capturing data on other axes, or A,B,C,D,E,F,G,H for SMC–4000.**

Example - Recording into An Array

During a position move, store the X and Y positions and position error every 2 msec.

#RECORD	Begin program
DM XPOS[300],YPOS[300]	Define X,Y position arrays

DM XERR[300],YERR[300]	Define X,Y error arrays
RA XPOS[],XERR[],YPOS[],YERR[]	Select arrays for capture
RD _TPX,_TEX,_TPY,_TEY	Select data types
PR 10000,20000	Specify move distance
RC1	Start recording now, at rate of 2 msec
BG XY	Begin motion
#A;JP #A,RC=1	Loop until done
MG "DONE"	Print message
EN	End program
#PLAY	Play back
N=0	Initial Counter
JP# DONE,N>300	Exit if done
N=	Print Counter
X POS[N]=	Print X position
Y POS[N]=	Print Y position
XERR[N]=	Print X error
YERR[N]=	Print Y error
N=N+1	Increment Counter
#DONE	Done

NOTES:

8 Input and Output of Data

Sending Messages

Messages may be sent to the bus using the MG (Message) command. This command sends specified text and numerical or string data from variables or arrays to the screen.

Text strings are specified in quotes and variable or array data is designated by the name of the variable or array. For formatting string variables, the {Sn} specifier is required where n is the number of characters, 1 through 6. Example:

MG STR {S3}

The above statement returns 3 characters of the string variable named STR.

Numeric data may be formatted using the {Fn.m} expression following the completed MG statement. {$n.m} formats data in HEX instead of decimal. Example:

MG "The Final Value is", RESULT {F5.2}

The above statement sends the message:

The Final Value is xxxxx.xx

The actual numerical value for the variable, RESULT, is substituted with the format of 5 digits to the left of the decimal and 2 to the right.

In addition to variables, functions and commands, responses can be used in the message command. For example:

MG "Analog input is", @AN[1]

MG "The Position of X is", _TPX

The message command normally sends a carriage return and line feed following the statement. The carriage return and the line feed may be suppressed by sending {N} at the end of the statement. This is useful when a text string needs to surround a numeric value.

Example:

#A	Label
JG 50000;BGX;ASX	Jog, Begin, After Speed
MG "The Speed is", _TVX {F5.1} {N}	Message
MG "counts/sec"	Message
EN	End Program

When #A is executed, the above example will appear on the screen as: The speed is 50000 counts/sec

The MG command can also be used to configure terminals. Here, any character can be sent by using {^n} where n is any integer between 1 and 255.

Example:

MG {^07} {^255} sends the ASCII characters represented by 7 and 255 to the bus.

Summary of Message Functions:

MG	Message command
" "	Surrounds text string
{Fn.m}	Formats numeric values in decimal n digits to the right of the decimal point and m digits to the left
{$n.m}	Formats numeric values in hexadecimal
{^n}	Sends ASCII character specified by integer n
{N}	Suppresses carriage return/line feed
{Sn}	Sends the first n characters of a string variable, where n is 1 through 6.

Variables may also be sent to the screen using the variable= format. Variable Name= returns the variable value. For example, V1= , returns the value of the variable V1.

Example - Printing a Variable

#DISPLAY	Label
PR 1000	Position Command
BGX	Begin
AMX	After Motion
V1=_TPX	Assign Variable V1
V1=	Print V1

Input of Data

The IN command is used to prompt the user to input numeric or string data. The input data is assigned to the specified variable or array element.

A message prompt may be sent to the user by specifying the message characters in quotes.

Example:

```
#A
IN "Enter Length", LENX
EN
```

This program sends the message:

```
Enter Length
```

to the PC screen or dumb terminal and waits for the operator to enter a value. The operator enters the numeric value which is assigned to the variable, LENX. String variables with up to six characters may also be input using the {S} specifier. For example, IN "Enter X,Y or Z", V{S} specifies a string variable to be input.

Formatting Data

Returned numeric values may be formatted in decimal or hexadecimal* with a specified number of digits to the right and left of the decimal point using the PF command.

The Position Format (PF) command formats motion values such as those returned by the Tell Position (TP), Speed? (SP?) and Tell Error (TE) commands.

Position Format is specified by:

PF m.n

where m is the number of digits to the left of the decimal point (0 through 10) and n is the number of digits to the right of the decimal point (0 through 4) A negative sign for m specifies hexadecimal format.

Hex values are returned preceded by a $ and in 2's complement. Hex values should be input as signed 2's complement, where negative numbers have a negative sign. The default format is PF 10.0.

Examples:

:DP21	Define position
:TPX	Tell position
0000000021	Default format
:PF4	Change format to 4 places
:TPX	Tell position
0021	New format
:PF-4	Change to hexadecimal format
:TPX	Tell Position
$0015	Hexadecimal value

The following interrogation commands are affected by the PF command:

DP
ER
PA
PR
TE
TP

If the number of decimal places specified by PF is less than the actual value, a nine appears in all the decimal places.

Example:

:PF2	Format 2 places
:TPX	Tell position
99	Returns 99 if actual position is more than allowed format

The Variable Format (VF) command is used to format variables and array elements. The VF command is specified by:

VF m.n

where m is the number of digits to the left of the decimal point (0 through 10) and n is the number of digits to the right of the decimal point (0 through 4).

A negative sign for m specifies hexadecimal format. The default format for VF is VF 10.4

Hex values are returned preceded by a $ and in 2's complement.

:V1=10	Assign V1
:V1=	Return V1
0000000010.0000	Default format
:VF2.2	Change format
:V1=	Return V1
10.00	New format
:VF-2.2	Specify hex format
:V1=	Return V1
$0A.00	Hex value
:VF1	Change format
:V1=	Return V1
9	Overflow

The variable format also affects returned values from internal variables such as _GNX.

PF and VF commands are global format commands. Parameters may also be formatted locally by using the {Fn.m} or {$n.m} specification following the variable = . For example:

V1={F4.2}	Specifies the variable V1 to be returned in a format of 4 digits to left of decimal and 2 to the right.

F specifies decimal and $ specifies hexadecimal. n is the number of digits to the left of the decimal, and m is the number of digits to the right of the decimal. The local format is used with the MG* command.

Examples:

:V1=10	Assign V1
:V1=	Return V1
0000000010.0000	Default Format
:V1={F4.2}	Specify local format
0010.00	New format
:V1={$4.2}	Specify hex format
$000A.00	Hex value
:V1="ALPHA"	Assign string "ALPHA" to V1
:V1={S4}	Specify string format first 4 characters
ALPH	

User Units

Variables and arithmetic operations make it easy to input data in desired user units i.e.; inches or RPM.

For example, an operator can be prompted to input a number in revolutions. The input number is converted into counts by multiplying it by the number of counts/revolution.

The SMC–4000 position parameters such as PR, PA and VP have units of quadrature counts. Speed parameters such as SP, JG and VS have units of counts/sec. Acceleration parameters such as AC, DC, VA and VD have units of counts/sec^2. All input parameters must be converted into these units.

Example:

#RUN	Label
IN "ENTER # OF REVOLUTIONS",N1	Prompt for revs
PR N1*2000	Convert to counts
IN "ENTER SPEED IN RPM",S1	Prompt for RPMs
SP S1*2000/60	Convert to counts/sec
IN "ENTER ACCEL IN RAD/SEC2",A1	Prompt for ACCEL
AC A1*2000/(2*3.14)	Convert to counts/sec2
BG	Begin motion
EN	End program

NOTES:

9 Programmable I/O

Digital Outputs

Each bit on the output port may be set and cleared with the software instructions SB (Set Bit) and CB(Clear Bit), or OB (define output bit).

Example:

Instruction	Function
SB3	Sets bit 3 of output port
CB4	Clears bit 4 of output port

The Output Bit (OB) instruction is useful for setting or clearing outputs depending on the value of a variable, array, input or expression. Any non-zero value results in a set bit.

Instruction	Function
OB1, POS	Set Output 1 if the variable POS is non-zero. Clear Output 1 if POS equals 0.
OB 2, @IN [1]	Set Output 2 if Input 1 is high. If Input 1 is low, clear Output 2.
OB 3, @IN [1]&@IN [2]	Set Output 3 only if Input 1 and Input 2 are high.
OB 4, COUNT [1]	Set Output 4 if element 1 in the array COUNT is non-zero.

The output port may also be written to as an 8-bit word using the instruction

OP (Output Port). This instruction allows a single command to define the state of the entire 8-bit output port, where 2^0 is output 1, 2^1 is output 2 and so on. A 1 designates that output is on.

Example:

Instruction	Function
OP6	Sets outputs 2 and 3 of output port to high. All other bits are 0. ($2^1 + 2^2 = 6$) (1100 binary)
OP0	Clears all bits of output port to zero
OP 15	Sets all bits of output port to one. ($2^0 + 2^1 + 2^2 + 2^3$) (1111 binary)

The output port is useful for firing relays or controlling external switches and events during a motion sequence.

Example - Turn ON Output After Move

#OUTPUT	Label
PR 2000	Position Command
BG	Begin
AM	After move
SB1	Set Output 1
WT 1000	Wait 1000 msec
CB1	Clear Output 1
EN	End

Digital Inputs

The SMC–4000 has eight digital inputs for controlling motion by local switches. The @IN[n] function returns the logic level of the specified input 1 through 8. For example, a Jump on Condition instruction can be used to execute a sequence if a high condition is noted on an input 3. To halt program execution, the After Input (AI) instruction waits until the specified input has occurred.

Example:

JP #A,@IN[1]=0	Jump to A if input 1 is low
JP #B,@IN[2]=1	Jump to B if input 2 is high
AI 7	Wait until input 7 is high
AI -6	Wait until input 6 is low

10 Example Applications

Instruction Set Examples

Below are some examples of simple instructions. It is assumed your system is hooked-up and the motors are under stable servo control.

DP*=0 <enter>	Define all axis positions as 0
PF 6,6,6,6 <enter>	Define position format as 6 digits
PR 100,200,300,400 <enter>	Specify X,Y,Z,W position command
BG <enter>	Begin Motion
TP <enter>	Tell Position
00100,00200,00300,00400	Returned Position data
PR?,?,?,? <enter>	Request Position Command
00100,00200,00300,00400	Returned data
BGX <enter>	Begin X axis only
TPX <enter>	Tell X position only
00200	Returned position data
tpx <enter>	Enter invalid command
? TC1 <enter>	Controller response - Request error code
1 Unrecognized command	Controller response
VM XY <enter>	Specify Vector Mode for XY
VS 10000 <enter>	Specify Vector Speed
VP 2000,3000 <enter>	Specify Vector Segment
VP 4000,5000 <enter>	Specify Vector
LE <enter>	Segment End Vector
BGS <enter>	Begin Coordinated Sequence
TPXY <enter>	Tell X and Y position
004200,005200	Returned data

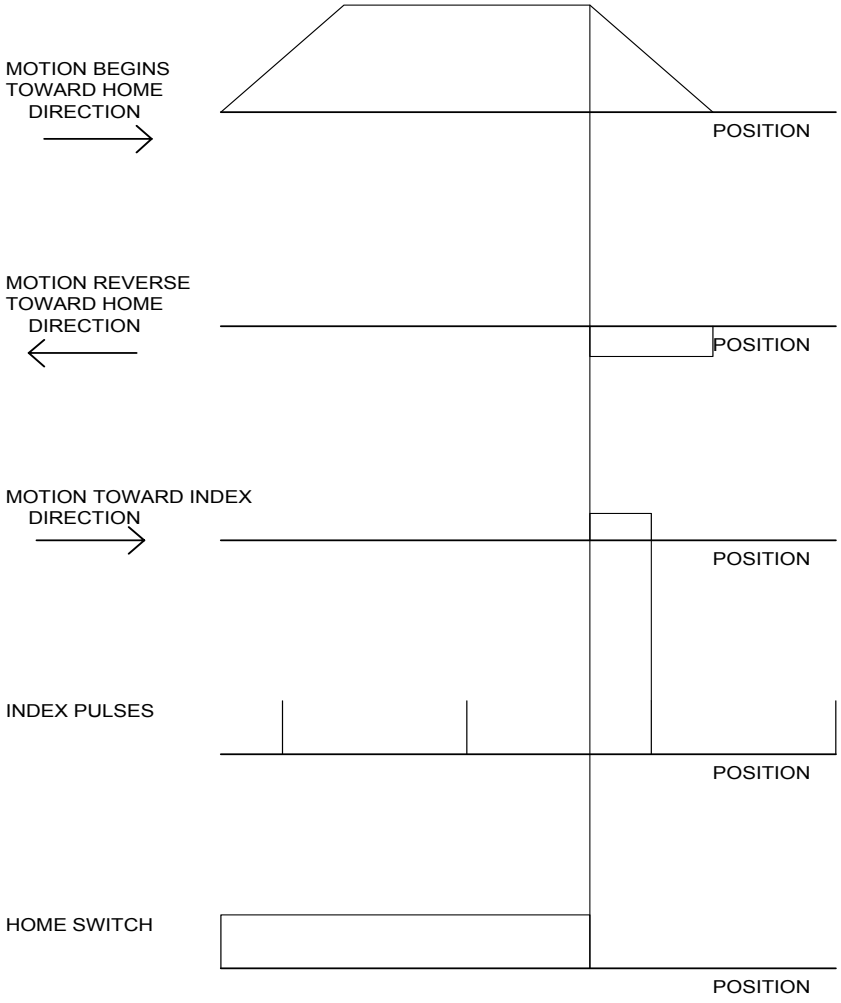
Example - Jog in X only

Jog X motor at 50000 count/s. After X motor is at its jog speed, begin jogging Z in reverse direction at 25000 count/s.

Instruction	Interpretation
#A	Label
AC 20000,20000	Specify X,Y acceleration of 20000 cts / sec
DC 20000,20000	Specify X,Y deceleration of 20000 cts / sec
JG 50000,-25000	Specify jog speed and direction for X and Y axis
BG X	Begin X motion
AS X	Wait until X is at speed
BG Z	Begin Z motion
EN	

Homing Example (HM method)

Instruction	Interpretation
#HOME	Label
AC 1000000	Acceleration Rate
DC 1000000	Deceleration Rate
SP 5000	Speed for Home Search
HM X	Home X
BG X	Begin Motion
AM X	After Complete
MG "AT HOME"	Send Message
EN	End
#EDGE	Label
AC 2000000	Acceleration rate
DC 2000000	Deceleration rate
SP 8000	Speed
FE Y	Find edge command
BG Y	Begin motion
AM Y	After complete
MG "FOUND HOME"	Send message
DP,0	Define position as 0
EN	End



Motion intervals in the Home sequence

Homing Example (FE and FI method)

This example demonstrates how to home servos with a home sensor in the middle of a slide where it is possible for the servo to be on either side of the home sensor at power-up. If the servo is already past the sensor, it will hit a limit switch first, and the #LIMSWI special label subroutine will reverse the CN (Configure Limit Switches) command and turn the servo around. The #BACKUP subroutine is used to make the servo come back to the home input and go a small distance past it, so the #HOMING routine can always hit the same side of the home sensor.

Ideally, the home sensor is a photo device. If there is a white and black strip along the slide, the photo eye will see either light or dark, and the value of _HMX will be “1” or “0”. Under this design, the FEX command can automatically determine the direction to find the transition point of the black and white strip. You need not account for the limit switch or the #BACKUP routine in that case

Instruction	Interpretation
#TEST	
SPX=10000	
ACX=1000000	
DCX=1000000	
CN,1	
TRUE=1	
FALSE=0	
HOMING=FALSE	
SHX; WT 2000	
#HOMING	
MG “Attempting to find home” {N}	
MG “(normal direction)”	
FLX	
BLX	
HOMING=TRUE	
JGX=8192; FEX;BGX;AMX	
HOMING=FALSE	
JG*=500; FIX; BGX; MCX	(MCX because the controller will automatically define the position as zero when the index is found)
MG “Homed O.K!” EN	
#BACKUP	
MG “Going back to the {N}	
MG “home input..”	
CN,-1	

Instruction	Interpretation
FEX; BGX; AMX	
IPX=-20000; AMX	(Need to adjust number based on distance)
CN,1	
JP #HOMING	
EN	
#LIMSWI; AB1	(AB1 optional, to instantly stop all servos)
JP#REALPRB,HOMING =FALSE	(Do next part if limit during homing)
zs;WT 1000; JP #BACKUP	(Next part handles a real limit event)
#REALPRB	
MG “Limit Hit”	
RE1	
#CMDERR	(Command error handler special label)
AB1; ZS	
JP#LIMSWI,_TC=22	(Refer to #LIMSWI handler if try to begin or motor OFF)
MG “Error”{N};TCI{N}	
MG”on line”,_ED{F3.0}	
MG”Program halted!”	
AB	
EN	

Example - Input Interrupt

Instruction	Interpretation
#A	Label #A
II 1	Enable input 1 for interrupt function
JG 30000,-20000	Set speeds on A and B axes
BG AB	Begin motion on A and B axes
#B	Label #B
TP AB	Report A and B axes positions
WI 1000	Wait 1000 milliseconds
JP #B	Jump to #B
EN	End of program
#INNT	Interrupt subroutine
MG “Interrupt has occured”	Displays the message
ST AB	Stops motion on A and B axes
#LOOP;JP#LOOP,@IN[1]=0	Loop until Interrupt cleared
JG 15000,10000	Specify new speeds
WT300	Wait 300 milliseconds
BG AB	Begin motion on A and B axes
RI	Return from Interrupt subroutine

Example - Position Follower (Point-to-Point)

Objective - The motor must follow an analog signal. When the analog signal varies by 10V, motor must move 10000 counts.

Method: Read the analog input and command X to move to that point.

Instruction	Interpretation
#POINTS	Label
SP 7000	Speed
AC 80000;DC 80000	Acceleration
#LOOP	Label
COMPP=@AN[1]*1000	Read analog input, and compute position
PA COMPP	Command position
BGX	Start motion
AMX	After completion
JP #LOOP	Repeat
EN	End

Example - Position Follower (Continuous Move)

Method: Read the analog input, compute the commanded position and the position error. Command the motor to run at a speed in proportions to the position error.

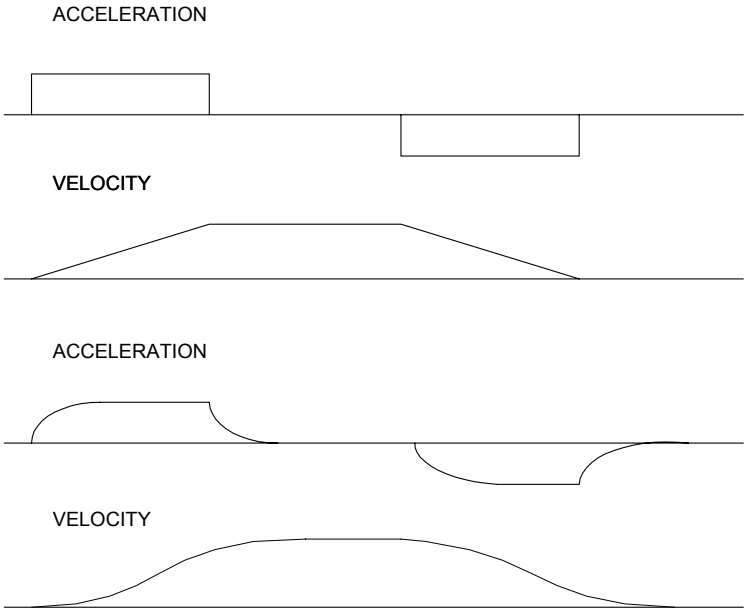
Instruction	Interpretation
#CONT	Label
AC 80000;DC 80000	Acceleration rate
JG 0	Start job mode
BGX	Start motion
#LOOP	
COMPP=@AN[1]*1000	Compute desired position
VE=COMPP-_TPX	Find position error
PVEL=VE*20	Compute velocity
JG PVEL	Change velocity
JP #LOOP	Repeat
EN	End

Example - Absolute Position Movement

PA 10000,20000	Specify absolute X,Y position
AC 1000000,1000000	Acceleration for X,Y
DC 1000000,1000000	Deceleration for X,Y
SP 50000,30000	Speeds for X,Y
BG XY	Begin motion

Example - Motion Smoothing

Instruction	Interpretation
PR 20000	Position
AC 100000	Acceleration
DC 100000	Deceleration
SP 5000	Speed
IT .5	Filter for S-curve
BG X	Begin



Trapezoidal Velocity and Smooth Velocity Profiles

Cut-to-Length Example

In this example, a length of material is to be advanced a specified distance. When the motion is complete, a cutting head is activated to cut the material. The length is variable, and the operator is prompted to input it in inches. Motion starts with a start button which is connected to input 1.

The load is coupled with a 2 pitch lead screw. A 2000 count/rev encoder is on the motor, resulting in a resolution of 4000 counts/inch. The program below uses the variable LEN, to length. The IN command is used to prompt the operator to enter the length, and the entered value is assigned to the variable LEN.

#BEGIN	Label
AC 800000	Acceleration
DC 800000	Deceleration
SP 5000	Speed
LEN=3.4	Initial length in inches
#CUT	Cut routine
AI1	Wait for start signal
IN "enter Length(IN)", LEN	Prompt operator for length in inches
PR LEN *4000	Specify position in counts
BGX	Begin motion to move material
AMX	Wait for motion done
SB1	Set output to cut
WT100;CB1	Wait 100 msec, then turn off cutter
JP #CUT	Repeat process
EN	End program

Latch Capture Example

Instruction	**Interpretation**
#Latch	Latch program
JG5000	Jog X
BG X	Begin motion on X axis
AL X	Arm Latch for X axis
#Wait	#Wait label for loop
JP #Wait,_ALX=1	Jump to #Wait label if latch has not occurred
Result=_RLX	Set value of variable ‘Result’ equal to the report position of X axis
Result=	Print result
EN	End

Example - Electronic Gearing SMC–4000

Objective: Run a geared motor at a speed of 1.132 times the speed of an external master. The master is driven at speeds between 0 and 1800 RPM (2000 counts/rev encoder), and is connected through the auxiliary encoder inputs.

Solution: Use an SMC–4000 controller, where the X-axis auxiliary is master and X-axis main is geared axis.

GR 1.132	Specify gear ratio

Now suppose the gear ratio of the X-axis is to change on-the-fly to 2. This can be achieved by commanding:

GR 2	Specify gear ratio for X axis to be 2

Contour Mode Example

A complete program to generate the contour movement in this example is given below. To generate an array, compute the position value at intervals of 8 ms. This is stored at the array POS. Then, the difference between the positions is computed and is stored in the array DIF. Finally the motors are run in the contour mode.

Instruction	Interpretation
#POINTS	Program defines X points
DM POS[16]	Allocate memory
DM DIF[15]	
C=0	Set initial conditions, C is index
T=0	T is time in ms
#A	
V1=50*T	
V2=3*T	Argument in degrees
V3=-955*@SIN[V2]+V1	Compute position
V4=@INT[V3]	Integer value of V3
POS[C]=V4	Store in array POS
T=T+8	
C=C+1	
JP #A,C<16	
#B	Program to find position differences
C=0	

Instruction	Interpretation
#C	
D=C+1	
DIF[C]=POS[D]-POS[C]	Compute the difference and store
C=C+1	
JP #C,C<15	
EN	End first program
#RUN	Program to run motor
CMX	Contour Mode
DT3	4 millisecond intervals
C=0	
#E	
CD DIF[C]	Contour Distance is in DIF
WC	Wait for completion
C=C+1	
JP #E, C<15	
DT0	
CD0	Stop Contour
EN	End the program

Example of Linear Interpolation Motion

Instruction	Interpretation
#LMOVE	Label
DP 0,0	Define position of X and Y axes to be 0
LMX	Define linear mode between X and Y axes.
LI 5000	Specify first linear segment
LI 0	Specify second linear segment
LE	End linear segments
VS 4000	Specify vector speed
BGS	Begin motion sequence
AV 4000	Set trippoint to wait until vector distance of 4000 is reached
VS 1000	Change vector speed
AV 5000	Set trippoint to wait until vector distance of 5000 is reached
VS 4000	Change vector speed
EN	Program end

Generating an Array

Consider the velocity and position profiles shown in the following illustration -*Velocity Profile with Sinusoidal Acceleration*. The objective is to rotate a motor a distance of 6000 counts in 120 ms. The velocity profile is sinusoidal to reduce the jerk and the system vibration. If we describe the position displacement in terms of A counts in B milliseconds, we can describe the motion in the following manner:

$\omega = (A/B) [1 - \cos (2\pi T/B)]$

$\mathbf{X} = (AT/B) - (A/2\pi)\sin (2\pi T/B)$

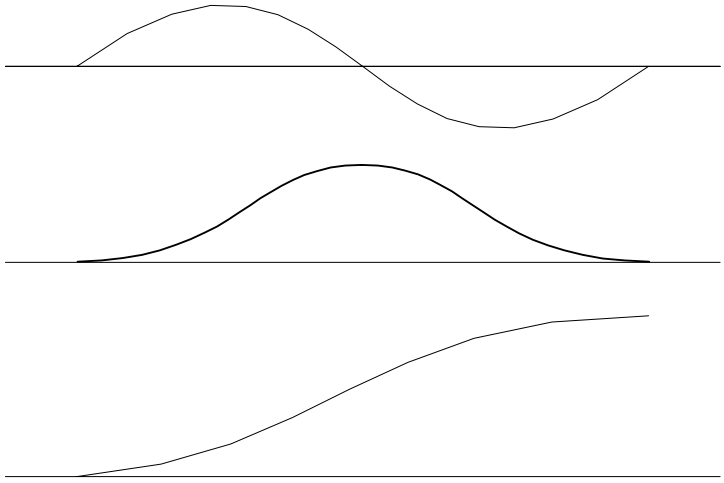
***Note ω is the angular velocity; X is the position; and T is the variable, time, in milliseconds.**

In the given example, A=6000 and B=120, the position and velocity profiles are:

$\mathbf{X} = 50T - (6000/2\pi) \sin (2\pi T/120)$

Note that the velocity, ω, in count/ms, is

$\omega = 50 [1 - \cos 2\pi T/120]$



Velocity Profile with Sinusoidal Acceleration

The 300 can compute trigonometric functions. However, the argument must be expressed in degrees. Using our example, the equation for X is written as:

X = 50T - 955 sin 3T

Teach (Record and Play-Back)

Several applications require a machine motion trajectory. Use SMC–4000 automatic array to capture position data. Captured data may be played back in contour mode. Use the following array commands:

DM C[n]	Dimension array
RA C[ ]	Specify array for automatic record (up to 4 for SMC–4000)
RD _TPX	Specify data for capturing (such as _TPX or _TPZ)
RC n,m	Specify capture time interval where n is 2n msec, m is number of records to be captured
RC? or _RC	Returns a 1 if recording

Record and Playback Example

Instruction	Interpretation
#RECORD	Begin Program
DM XPOS[501]	Dimension array with 501 elements
RA XPOS[]	Specify automatic record
RD _TPX	Specify X position to be captured
MOX	Turn X motor off
RC2	Begin recording; 4 msec interval
#A;JP#A,_RC=1	Continue until done recording
#COMPUTE	Compute DX
DM DX[500]	Dimension Array for DX
C=0	Initialize counter
#L	Label
D=C+1	
DELTA=XPOS[D]-XPOS[C]	Compute the difference
DX[C]=DELTA	Store difference in array
C=C+1	Increment index
JP #L,C<500	Repeat until done
#PLAYBCK	Begin Playback
CMX	Specify contour mode
DT2	Specify time increment
I=0	Initialize array counter
#B	Loop counter
CD XPOS[I];WC	Specify contour data I=I+1 Increment array counter JP #B,I<500 Loop until done
DT 0;CD0	End contour mode
EN	End program

Example - Multiple Move Sequence

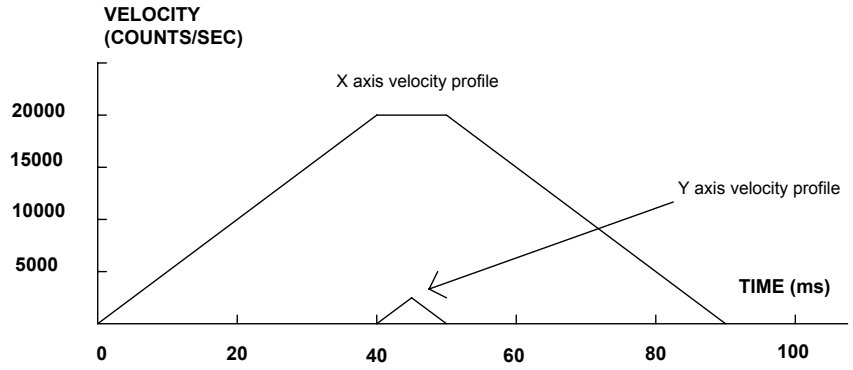
Required Motion Profiles:

X-Axis	2000 counts	Position	Y-Axis	100 counts	Position
	15000 count/sec	Speed		5000 count/sec	Speed
	500000 counts/sec^2	Acceleration		500000 counts/sec^2	Acceleration

This specifies relative position movement on the X axis. The movement is separated by 40 msec.

The following illustration - *Velocity Profiles of XY* shows the velocity profiles for the X and Y axis.

Instruction	Interpretation
#A	Begin Program
PR 2000,100	Specify relative position movement of 2000 and 100 counts for the X and Y axes.
SP 15000,5000	Specify speed of 15000 and 5000 counts / sec
AC 500000,500000	Specify acceleration of 500000 counts / sec^2 for all axes
DC 500000,500000	Specify deceleration of 500000 counts / sec^2 for all axes
BG X	Begin motion on the X axis
WT 40	Wait 40 msec
BG Y	Begin motion on the Y axis
EN	End Program



Velocity Profiles of XY

Notes on *Velocity Profiles of XY* illustration: The X axis has a 'trapezoidal' velocity profile, while the Y axis has a 'triangular' velocity profile. The X axis accelerates to the specified speed, moves at this constant speed, and then decelerates such that the final position agrees with the commanded position, PR. The Y axis accelerates, but before the specified speed is achieved, must begin deceleration such that the axis will stop at the commanded position.

Example - Start Motion on Switch

Motor X must turn at 4000 counts/sec when the user flips a panel switch to on. When panel switch is turned to off position, motor X must stop turning.

Solution: Connect panel switch to input 1 of SMC–4000. High on input 1 means switch is ON.

Instruction	Interpretation
#S;JG 4000	Set speed
AI 1;BGX	Begin after input 1 goes high
AI -1;STX	Stop after input 1 goes low
AMX;JP #S	After motion, repeat
EN;	

Examples - Input Interrupt

#A	Label #A
II 1	Enable input 1 for interrupt function
JG 30000,-20000	Set speeds on X and Y axes
BG XY	Begin motion on X and Y axes
#B	Label #B
TP XY	Report X and Y axes positions
WT 1000	Wait 1000 milliseconds
JP #B	Jump to #B
EN	End of program
#ININT	Interrupt subroutine
MG "Interrupt has occurred"	Displays the message
ST XY	Stops motion on X and Y axes
#LOOP;JP #LOOP,@IN[1]=0	Loop until Interrupt cleared
JG 15000,10000	Specify new speeds
WT 300	Wait 300 milliseconds
BG XY	Begin motion on X and Y axes
RI	Return from Interrupt subroutine

Special Labels

This program demonstrates five of the SPECIAL LABELS as part of an SMC–4000 application program. #AUTO is usually the first line of a program. When this program is burned into the SMC–4000 using the BP command, the program will begin executing when the power is turned ON or after the RS command is given, or the RESET button on the front is pressed.

Instruction
#AUTO
ERX=150; OEX=1, II3
SHX WT 500
#BUSY
JGX=@AN[1]*10000
BGX
MG “BUSY...”
WT500
JP#BUSY
EN

#POSERR-- This special label is used to handle a situation in which a servo is not able to remain in position. The special label works with the ER command. When the value of the ER command is exceeded, thread zero automatically jumps to the #POSERR label. In this program example, ERX=150 counts. If there are low gains or if using a small motor, it is possible to cause more than 150 counts of error by hand, causing the #POSERR label to execute. In the following example, the program displays a message and waits for input #1 to go low (falling edge). The servo is then re-energized.

There are three ways to return from a special label like this. The example below uses RE1; i.e., to return from the error routine to the line in thread zero that was being executed when the #POSERR occurred. The “1” means to restore a trip point if one was in progress, such as WT, AI, AM, AT, etc.

The second method is to do an RE, meaning that any trip points that were in progress are cleared. If thread zero was waiting for an AM command, it would continue as if the profiler had completed the path.

The third method is to use the ZS command, which clears the subroutine stack, and the LEGEND forgets it is in the middle of an error routine. After the ZS is given, it is possible to do a JP anywhere in the program. Typically, there would be a jump back to a main loop where manual jogging can take place.

Instruction
#POSERR
SB1
MG “FOLLOWING ERROR IS HIGH!”
MG “TOGGLE INPUT #1 TO CONTINUE”
AI1; AI-1
CB1; SHX; WT 500
RE1

The following is the special label that is automatically executed when there is a programming error, a command given where it cannot be used, or a number out of range for a command. The example below includes a jump to the #LIMSWI label if the _TC code is 22, which is “Begin not valid due to limit switch.” This is considered a command error, but is easier to treat as a limit switch error. Similar conditions could be handled by checking other _TC codes and reacting accordingly. If the error is anything other than 22, motion is aborted without aborting the program (AB1), then a message is prompted indicating the type of error and the line number on which it occurred. _ED reports the last line that had an error. The #CMDERR routine can be finished just like the #POSERR special label, but it is not recommended because usually there is very little reason to continue execution of the program if there are serious errors in it. This routine is very useful in two ways:

First, during program design when there will be many programming mistakes, it is convenient to have the program display the error and line number automatically.

Second, it is safer to abort motion if there is a program fault. Without the AB1command, the motors will continue doing whatever they were doing before the fault. For example, if they were jogging, they will continue jogging.

Instruction
#CMDERR
JP#LIMSWI,_TC=22
AB1
MG “Error”{N};TCI{N}
MG “on line”,_ED{F3.0}
MG “Program halted!”
AB
EN

The following is the #LIMSWI special label for handling situations where limit switches ar hit during motion. This label automatically executes if an axis is in motion and a limit switch in the direction of motion is hit, or a software limit is exceeded. Without this special label, if a limit switch is hit during motion, such as a position absolute move, the motor will decelerate to a stop with NO ERROR. if an AM command is used, it will be cleared. The example as shown does not recover from the limit switch error, but a recovery method that woks well is the use of a status flag variable. For example if the machine was in a manual jog operation, a variable could be used to indicate that it was in jog mode (JOGMODE=1). The first line in the #LIMSWI could jump to #PROBLEM if JOGMODE <>1, otherwise return from the error. The two commented lines below demonstrate this. (The JOGMODE variable can be set to “1” in the jog routine and set back to “0” at the end of the jog routine.)

Instruction
Limit= “+”
Axis=”X”; JS #HARD,_LFX=0; JS#SOFT,_FLX<_TPX
Axis=”Y”; JS #HARD,_LFY=0; JS#SOFT,_FLY<_TPY
Limit=“-”
Axis=“X”; JS#HARD,_LRX=0; JS#SOFT, _BLX>_TPX
Axis=“Y”; JS#HARD,_LRY=0; JS#SOFT, _BLY>_TPY (JP#PROBLEM,JOGMODE=0;REI) (#PROBLEM)
AB1; HX1; HX2; HX3
ZS
MG “PROGRAM HALTED! (LIMSWI)”
EN
#HARD;MG Limit {S}, “,Axis,“HARDWARE LIMIT HIT!”;EN
#SOFT;MG Limit {S}, “,Axis,“SOFTWARE LIMIT HIT!”;EN

The following is the special label to handle input interrupts. Inputs 1 - 8 can be used as interrupts. this example uses the input to tell the LEGEND that the system is under an E-STOP condition. This input may come from a contact that also removes power from the amplifiers. Notice that the interrupt command II is used at the beginning of the program to designate input #3 as an interrupt. When this input goes low, thread zero automatically jumps to #ININT if it is included in the program. Notice that if the example assumes that if an E-STOP occurs, the current operation has been scrapped. The ZS (Zero Subroutine Stack) command is used which allows the program to jump anywhere. Usually it is easiest to jump back to a main loop which handles the different modes of operation of the machine. Also note that if ZS is used, the interrupt must be enabled for next time.

Instruction	Interpretation
#ININT	
AB1; HX1; HX2; HX3	
SB3	
MG “ESTOPPED”	
AI-3; AI3	(Wait for e-stop input to go high (re-enabled))
CB3	
MG “RE-ENABLED..”	
SHX	
WT2000	
ZS	
II3	(Re-enable input interrupt for next time)
#JP BUSY	
EN	

Wire Cutter

Activate the start switch. The motor will advance the wire a distance of 10". When motion stops, the controller generates an output signal activating the cutter. Allow 100 ms for cutting to complete the cycle.

Suppose the motor drives the wire by a roller with a 2" diameter and the encoder resolution is 1000 lines per revolution. Since the circumference of the roller equals 2π inches, and it corresponds to 4000 quadrature, one inch of travel equals: 4000/2π = 637 count/inch

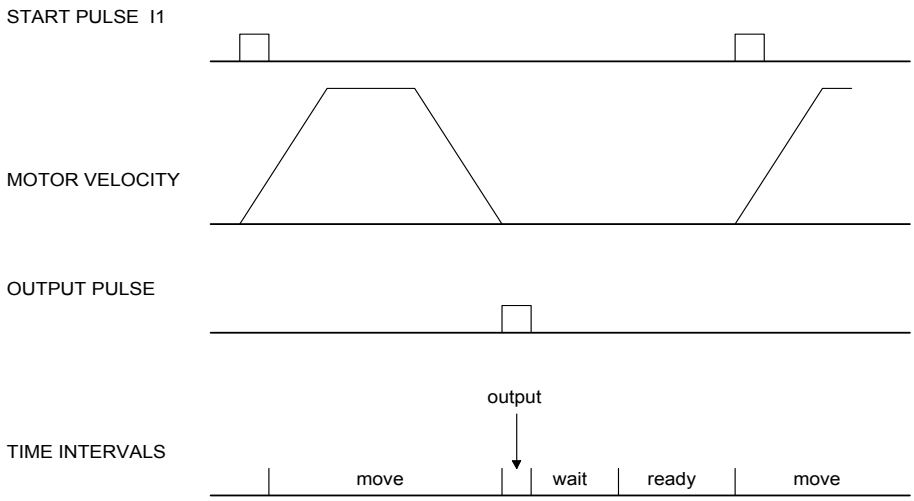
A distance of 10 inches equals 6370 counts, and a slew speed of 5 inches / second equals 3185 count/sec.

The input signal may be applied to I1, and the output signal as output 1. Motor velocity profile and related input and output signals are in the following illustration - *Motor Velocity and the Associated Input/Output Signals*.

The program starts at a state that we define as #A. Here the controller waits for the input pulse on I1. As soon as the pulse is given, the controller starts the forward motion.

Upon completion of the forward move, the controller outputs a pulse for 20 ms and then waits an additional 80 ms before returning to #A for a new cycle.

Instruction	Function
#A	Label
AI1	Wait for input 1
PR 6370	Distance
SP 3185	Speed
BGX	Start Motion
AMX	After motion is complete
SB1	Set output bit 1
WT 20	Wait 20 ms
CB1	Clear output bit 1
WT 80	Wait 80 ms
JP #A	Repeat the process



Motor Velocity and the Associated Input/Output Signals

Speed Control by Joystick

The speed of a motor is controlled by a joystick. The joystick produces a signal in the range between -10V and +10V. The objective is to drive the motor at a speed proportional to the input voltage.

Assume that a full voltage of 10 Volts must produce a motor speed of 3000 rpm with an encoder resolution of 1000 lines or 4000 count/rev. This speed equals:

3000 rpm = 50 rev/sec = 200000 count/sec

The program reads the input voltage periodically and assigns its value to the variable VIN. To get a speed of 200,000 ct/sec for 10 volts, we select the speed as

Speed = 20000 x VIN

The corresponding velocity for the motor is assigned to the VEL variable.

Instruction	Function
#A	Label
JG0	Set motor in jog mode speed zero
BGX	Start motion
#B	Label
VIN=@AN[1]	Read analog input
VEL=VIN*20000	Compute the desired velocity
JG VEL	Change the jog speed
JP #B	Repeat the process
EN	End

Position Control by Joystick

This system requires the position of the motor to be proportional to the joystick angle. Furthermore, the ratio between the two positions must be programmable. For example, if the control ratio is 5:1, it implies that when the joystick voltage is 5 Volts, corresponding to 4095 counts, the required motor position must be 20475 counts. The variable V3 changes the position ratio.

Instruction	Function
#A	Label
V3=5	Initial position ratio
DP0	Define the starting position
JG0	Set motor in jog mode as zero
BGX	Start
#B	
V1=@AN[1]	Read analog input
V2=V1*V3	Compute the desired position
V4=V2-_TPX-_TEX	Find the following error

Instruction	Function
V5=V4*20	Compute a proportional speed
JG V5	Change the speed
JP #B	Repeat the process
EN	End

Backlash Compensation by Dual-Loop

This design example addresses the basic problems of backlash in motion control systems. The objective is to control the position of a linear slide precisely. The slide is to be controlled by a rotary motor, which is coupled to the slide by a leadscrew. Such a leadscrew has a backlash of 4 micron, and the required position accuracy is for 0.5 micron.

The dilemma is where to mount the sensor. A rotary sensor, gives a 4-micron backlash error. If a linear encoder is used, the backlash in the feedback loop will cause oscillations due to instability.

An alternative approach is the dual-loop, using two sensors, rotary and linear. The rotary sensor assures stability (because the position loop is closed before the backlash); the linear sensor provides accurate load position information. The principle is to drive the motor to a given rotary position near the final point. The load position is then read to find position error and the controller commands the motor to move to a new rotary position which eliminates the position error.

Since the required accuracy is 0.5 micron, the resolution of the linear sensor should preferably be twice finer. A linear sensor with a resolution of 0.25 micron allows a position error of +/-2 counts.

The dual-loop approach requires the resolution of the rotary sensor to be equal or better than that of the linear system. Assuming that the pitch of the lead screw is 2.5mm (approximately 10 turns per inch), a rotary encoder of 2500 lines per turn or 10,000 count per revolution results in a rotary resolution of 0.25 micron. This results in equal resolution on both linear and rotary sensors.

To illustrate the control method, assume that the rotary encoder is used as a feedback for the X-axis, and that the linear sensor is read and stored in the variable LINPOS. Further assume that at the start, both the position of X and the value of LINPOS are equal to zero. Now assume that the objective is to move the linear load to the position of 1000.

The first step is to command the X motor to move to the rotary position of 1000. Once it arrives we check the position of the load. If, for example, the load position is 980 counts, it implies that a correction of 20 counts must be made. However, when the X-axis is commanded to be at the position of 1000, suppose that the actual position is only 995, implying that X has a position error of 5 counts, which will be eliminated once the motor settles. This implies that the correction needs to be only 15 counts, since 5 counts out of the 20 would be corrected by the X-axis. Accordingly, the motion correction should be:

Correction = Load Position Error - Rotary Position Error

The correction can be performed a few times until the error drops below +/-2 counts. Often, this is performed in one correction cycle.

Example Motion Program:

Instruction	Function
#A	Label
DP0	Define starting positions as zero
LINPOS=0	
PR 1000	Required distance
BGX	Start motion
#B	
AMX	Wait for completion
WT 50	Wait 50 msec
LIN POS = _DEX	Read linear position
ER=1000-LINPOS-_TEX	Find the correction
JP #C,@ABS[ER]<2	Exit if error is small
PR ER	Command correction
BGX	
JP #B	Repeat the process
#C	
EN	

11 Troubleshooting

Overview

The following discussion may help you get your system running if a problem is encountered.

Potential problems have been divided into groups as follows:

1. Installation
2. Stability and Compensation
3. Operation

The various symptoms along with the cause and the remedy are described in the following tables.

Installation

Symptom	Cause	Remedy
Motor runs away when connected to amplifier with no additional inputs.	Amplifier offset too large.	Adjust amplifier offset
Same as above, but offset adjustment does not stop motor.	Damaged amplifier.	Replace amplifier.
Controller does not read changes in encoder position.	Wrong encoder connections.	Check encoder wiring.
Same as above	Bad encoder	Check the encoder signals. Replace encoder if necessary.
Same as above	Bad controller	Connect the encoder to different axis input. If it works, controller failure. Repair or replace.

Stability

Symptom	Cause	Remedy
Motor runs away when the loop is closed.	Wrong feedback polarity. (Positive Feedback)	Invert the polarity of the loop by inverting the motor leads (brush type) or the encoder (channel A+, B+ if single ended; channel A+, A- and B+, B- if differential)
Motor oscillates.	Too high gain or too little damping.	Decrease KI and KP. Increase KD.

Operation

Symptom	Cause	Remedy
Controller rejects command. Responded with a ?	Anything.	Interrogate the cause with TC or TC1.
Motor does not start or complete a move.	Noise on limit switches stops the motor. Noise on the abort line aborts the motion.	To check the cause, interrogate the stop code (SC). If caused by limit switch or abort line noise, reduce noise.
During a periodic operation, motor drifts slowly.	Encoder noise	Interrogate the position periodically. If controller states that the position is the same at different locations it implies encoder noise. Also use a scope to see the noise. Reduce noise. Use differential encoder inputs.
Same as above.	Programming error.	Avoid resetting position error at end of move with SH command.

12 Index

Symbols

A

B

C

D

E

M

N

O

YASKAWA ELECTRIC AMERICA, INC.
Chicago-Corporate Headquarters 2121 Norman Drive South, Waukegan, IL 60085, U.S.A.
Phone: (847) 887-7000 Fax: (847) 887-7310 Internet: http://www.yaskawa.com
MOTOMAN INC.
805 Liberty Lane, West Carrollton, OH 45449, U.S.A.
Phone: (937) 847-6200 Fax: (937) 847-6277 Internet: http://www.motoman.com
YASKAWA ELECTRIC CORPORATION
New Pier Takeshiba South Tower, 1-16-1, Kaigan, Minatoku, Tokyo, 105-0022, Japan
Phone: 81-3-5402-4511 Fax: 81-3-5402-4580 Internet: http://www.yaskawa.co.jp
YASKAWA ELETRICO DO BRASIL COMERCIO LTDA.
Avenida Fagundes Filho, 620 Bairro Saude Sao Paolo-SP, Brasil CEP: 04304-000
Phone: 55-11-5071-2552 Fax: 55-11-5581-8795 Internet: http://www.yaskawa.com.br
YASKAWA ELECTRIC EUROPE GmbH
Am Kronberger Hang 2, 65824 Schwalbach, Germany
Phone: 49-6196-569-300 Fax: 49-6196-888-301 Internet: http://www.yaskawa.de
MOTOMAN ROBOTICS AB
Box 504 S38525, Torsas, Sweden
Phone: 46-486-48800 Fax: 46-486-41410
MOTOMAN ROBOTEC GmbH
Kammerfeldstrabe 1, 85391 Allershausen, Germany
Phone: 49-8166-900 Fax: 49-8166-9039
YASKAWA ELECTRIC UK LTD.
1 Hunt Hill Orchardton Woods Cumbernauld, G68 9LF, Scotland, United Kingdom
Phone: 44-12-3673-5000 Fax: 44-12-3645-8182
YASKAWA ELECTRIC KOREA CORPORATION
Paik Nam Bldg. 901 188-3, 1-Ga Euljiro, Joong-Gu, Seoul, Korea
Phone: 82-2-776-7844 Fax: 82-2-753-2639
YASKAWA ELECTRIC (SINGAPORE) PTE. LTD.
Head Office: 151 Lorong Chuan, #04-01, New Tech Park Singapore 556741, SINGAPORE
Phone: 65-282-3003 Fax: 65-289-3003
TAIPEI OFFICE (AND YATEC ENGINEERING CORPORATION)
10F 146 Sung Chiang Road, Taipei, Taiwan
Phone: 886-2-2563-0010 Fax: 886-2-2567-4677
YASKAWA JASON (HK) COMPANY LIMITED
Rm. 2909-10, Hong Kong Plaza, 186-191 Connaught Road West, Hong Kong
Phone: 852-2803-2385 Fax: 852-2547-5773
BEIJING OFFICE
Room No. 301 Office Building of Beijing International Club,
21 Jianguomanwai Avenue, Beijing 100020, China
Phone: 86-10-6532-1850 Fax: 86-10-6532-1851
SHANGHAI OFFICE
27 Hui He Road Shanghai 200437 China
Phone: 86-21-6553-6600 Fax: 86-21-6531-4242
SHANGHAI YASKAWA-TONJI M & E CO., LTD.
27 Hui He Road Shanghai 200437 China
Phone: 86-21-6533-2828 Fax: 86-21-6553-6677
BEIJING YASKAWA BEIKE AUTOMATION ENGINEERING CO., LTD.
30 Xue Yuan Road, Haidian, Beijing 100083 China
Phone: 86-10-6232-9943 Fax: 86-10-6234-5002
SHOUGANG MOTOMAN ROBOT CO., LTD.
7, Yongchang-North Street, Beijing Economic & Technological Development Area,
Beijing 100076 China
Phone: 86-10-6788-0551 Fax: 86-10-6788-2878
YEA, TAICHUNG OFFICE IN TAIWAN
B1, 6F, No. 51, Section 2, Kung-Yi Road, Taichung City, Taiwan, R.O.C.
Phone: 886-4-2320-2227 Fax: 886-4-2320-2239
Phone: 55-11-5071-2552 Fax: 55-11-5581-8795 Internet: http://www.yaskawa.com.br