



# LEGEND-MC

## 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

<b>1 Introduction .....</b>	<b>1</b>
Part Numbers .....	2
Startup .....	3
Mounting the LEGEND-MC to the LEGEND Amplifier .....	3
Mounting Orientation .....	3
Front Panel Description.....	4
Power/Connections Wiring - Single Phase .....	5
Power/Connections Wiring - Three Phase .....	6
Cable Shielding, Segregation and Noise Immunity .....	7
I/O Connections (50-pin CN5).....	8
Analog I/O .....	9
Analog Input .....	9
Analog Output .....	10
Digital I/O .....	11
Digital Input .....	11
Digital Output .....	12
Emergency Stop Chain .....	13
Serial Communication .....	14
External Encoder Specifications .....	15
Dedicated Inputs .....	16
Physical Specifications .....	17
Hardware Specifications .....	17
Cable Diagram and Dimensional Drawings .....	18
I/O Cable with Terminal Block JUSP-TA50P .....	18
<b>2 Theory of Operation .....</b>	<b>21</b>
Overview .....	21
Level .....	22
Operation of Closed-Loop Systems .....	24
System Modeling .....	25
Controller .....	25
Motor-Amplifier .....	25
Current Drive .....	25
Velocity Loop .....	26
Encoder .....	27
DAC .....	27
Digital Filter .....	27
ZOH .....	28
System Analysis .....	28
System Design and Compensation .....	31
The Analytical Method .....	31
<b>3 Communications .....</b>	<b>35</b>
Introduction .....	35
Controller Response to Data .....	35
RS232 Port .....	35
Ethernet Configuration .....	36
Communication Protocols .....	36
Addressing .....	37
Ethernet Handles .....	37
Global vs Local Operation .....	38
Configuring Operation for Distributed Control .....	39
Operation of Distributed Control .....	40
Accessing the I/O of the slaves .....	40
Handling Communication Errors .....	41
Modbus Support .....	41

Communicating with Multiple Devices .....	43
Multicasting .....	43
Using Third Party Software .....	43
<b>4 Command Reference .....</b>	<b>45</b>
Command Description .....	49
AB Abort .....	51
@ABS Absolute Value Function .....	52
AC Acceleration .....	53
@ACOS Arc Cosine Function.....	54
AD After Distance .....	55
AF Analog Feedback .....	56
AI After Input .....	57
AL Arm Latch .....	58
AM After Move .....	59
@AN Read Analog Input.....	60
AO Analog Out .....	61
AP After Absolute Position .....	62
AR After Relative Distance .....	63
AS At Speed .....	64
@ASIN Arc Sine Function.....	65
AT At Time .....	66
@ATAN Arc Tangent Function .....	67
BG Begin.....	68
BL Reverse Software Limit .....	69
BN Burn .....	70
BP Burn Program .....	71
BV Burn Variables and Array .....	72
CB Clear Bit .....	73
CD Contour Data .....	74
CE Configure Encoder .....	75
CF Configure Messages .....	76
CH Connect Handle .....	77
CM Contouring Mode .....	79
CN Configure Limit Switches .....	80
@COM 2's Complement .....	81
@COS Cosine .....	82
CS Clear Sequence .....	83
CW Copyright information / Data Adjustment Bit On/Off .....	84
DA Deallocate Variables & Arrays .....	85
DC Deceleration .....	86
DE Dual (Auxiliary) Encoder Position .....	87
DL Download .....	88
DM Dimension .....	89
DP Define Position .....	90
DT Delta Time .....	91
DV Dual Velocity (Dual Loop) .....	92
EA ECAM master .....	93
EB ECAM Enable .....	94
EC ECAM Counter .....	95
ED Edit .....	96
EG ECAM Engage .....	97
ELSE Else Function for use with IF Conditional Statement .....	98
EM ECAM Cycle .....	99
EN End .....	100
ENDIF End of IF Conditional Statement .....	101
EO Echo .....	102

EP Cam Table Intervals and Starting Point .....	103
EQ ECAM Quit (disengage) .....	104
ER Error Limit .....	105
ET Electronic Cam Table .....	106
FA Acceleration Feedforward .....	107
FE Find Edge .....	108
FI Find Index .....	109
FL Forward Software Limit .....	110
@FRAC Fraction .....	111
FV Velocity Feedforward .....	112
GA Master Axis for Gearing .....	113
GR Gear Ratio .....	114
HM Home .....	115
HX Halt Execution .....	116
IA IP Address .....	117
IF IF conditional statement .....	118
IH Open Internet Handle .....	119
II Input Interrupt .....	121
IL Integrator Limit .....	123
IN Input Variable .....	124
@IN Status of Digital Input .....	125
@INT Integer .....	126
IP Increment Position .....	127
IT Independent Time Constant - Smoothing Function .....	128
JG Jog .....	129
JP Jump to Program Location .....	130
JS Jump to Subroutine .....	131
KD Derivative Constant .....	132
KI Integrator .....	133
KP Proportional Constant .....	134
LA List Arrays .....	135
LE Linear Interpolation End .....	136
_LF* Forward Limit Switch Operand (Keyword) .....	137
LI Linear Interpolation Distance .....	138
LL List Labels .....	140
LM Linear Interpolation Mode .....	141
LO Lockout .....	142
_LR* Reverse Limit Switch Operand (Keyword) .....	143
LS List Program .....	144
LV List Variables .....	145
LZ Inhibit Leading Zeros .....	146
MB Modbus .....	147
MC Motion Complete - "In Position" .....	149
MF Forward Motion to Position .....	150
MG Message .....	151
MM Master's Modulus .....	152
MO Motor Off .....	153
MR Reverse Motion to Position .....	154
MT Motor Type .....	155
NA Number of Axes .....	156
NB Notch Bandwidth .....	157
NF Notch Frequency .....	158
NO No Operation .....	159
NZ Notch Zero .....	160
OB Output Bit .....	161
OC Output Compare .....	162

OE Off on Error .....	163
OF Offset .....	164
OP Output Port .....	165
@OUT Status of Digital Output .....	166
PA Position Absolute .....	167
PF Position Format .....	168
PR Position Relative .....	170
QD Download Array .....	171
QR Data Record .....	172
QU Upload Array .....	173
QW Slave Record Update Rate .....	174
QZ Return Data Record Information .....	175
RA Record Array .....	176
RC Record .....	177
RD Record Data .....	178
RE Return from Error Routine .....	180
RI Return from Interrupt Routine .....	181
RL Report Latched Position .....	182
@RND Round .....	183
RS Reset .....	184
<control>R<control>S Master Reset .....	185
<control>R<control>V Revision .....	186
SA Send Command .....	187
SB Set Bit .....	188
SC Stop Code .....	189
SH Servo Here .....	190
@SIN Sin .....	191
SP Speed .....	192
@SQR Square Root .....	193
ST Stop .....	194
TB Tell Status Byte .....	195
TC Tell Error Code .....	196
TD Tell Dual Encoder .....	199
TE Tell Error .....	200
TI Tell Inputs .....	201
TIME* Time Operand (Keyword) .....	202
TL Torque Limit .....	203
TM Time .....	204
TP Tell Position .....	205
TR Trace .....	206
TS Tell Switches .....	207
TT Tell Torque .....	209
TV Tell Velocity .....	210
TW Timeout for IN-Position (MC) .....	211
UL Upload .....	212
VA Vector Acceleration .....	213
VD Vector Deceleration .....	214
VE Vector Sequence End .....	215
VF Variable Format .....	216
VR Vector Speed Ratio .....	217
VS Vector Speed .....	218
VT Vector Time Constant - S curve .....	219
WC Wait for Contour Data .....	220
WT Wait .....	221
XQ Execute Program .....	222
ZA User Variable .....	223

ZB User Variable .....	224
ZS Zero Subroutine Stack .....	225
COMMAND INTERROGATION LIST.....	226
<b>5 Programming Basics .....</b>	<b>229</b>
Introduction .....	229
Program Maximums .....	229
Command Syntax .....	229
Controller Response to Commands .....	231
Command Summary .....	232
Motion .....	232
Program Flow .....	234
General Configuration .....	235
Control Filter Settings .....	236
Status .....	237
Error And Limits .....	237
Arithmetic Functions .....	238
<b>6 Programming Motion .....</b>	<b>239</b>
Overview .....	239
Independent Axis Positioning .....	241
Command Summary - Independent Axis .....	241
Independent Jogging .....	243
Command Summary - Jogging .....	243
Linear Interpolation Mode .....	244
Specifying Linear Segments .....	244
Command Summary - Linear Interpolation .....	246
Vector Mode: Linear Interpolation Motion .....	248
Specifying Vector Segments .....	248
Additional Commands .....	248
Command Summary - Coordinated Motion Sequence .....	249
Operand Summary - Coordinated Motion Sequence .....	250
Electronic Gearing .....	251
Command Summary - Electronic Gearing .....	251
Electronic Cam .....	252
Contour Mode .....	256
Specifying Contour Segments .....	256
Additional Commands .....	257
Command Summary - Contour Mode .....	257
General Velocity Profiles.....	257
Motion Smoothing .....	258
Using the IT and VT Commands (S curve profiling).....	258
Homing.....	259
High Speed Position Capture (Latch Function) .....	260
<b>7 Application Programming .....</b>	<b>261</b>
Introduction .....	261
Program Format .....	261
Special Labels .....	262
Executing Programs - Multitasking .....	263
Debugging Programs .....	264
Program Flow Commands .....	266
Event Triggers & Trippoints .....	267
LEGEND-MC Event Triggers .....	267
Event Trigger Examples .....	268
Conditional Jumps .....	272
Multiple Conditional Statements .....	274
If, Else, and Endif .....	276
Command Format - IF, ELSE and ENDIF .....	277

Subroutines .....	278
Stack Manipulation .....	278
Auto Start Routine .....	278
Automatic Subroutines for Monitoring Conditions .....	279
Mathematical and Functional Expressions .....	282
Variables .....	284
Programmable Variables .....	284
Internal Variables & Keywords .....	286
Arrays.....	288
Defining Arrays .....	288
Assignment of Array Entries .....	288
Automatic Data Capture into Arrays .....	290
<b>8 Input and Output of Data .....</b>	<b>293</b>
Sending Messages .....	293
Input of Data .....	295
Formatting Data .....	295
User Units .....	297
<b>9 Programmable I/O .....</b>	<b>299</b>
Digital Outputs .....	299
Digital Inputs .....	300
Input Interrupt Function .....	300
Analog Inputs .....	300
<b>10 Example Applications .....</b>	<b>301</b>
Instruction Set Examples .....	301
Special Labels .....	318
Wire Cutter .....	322
Speed Control by Joystick .....	323
Position Control by Joystick .....	324
Backlash Compensation by Dual-Loop .....	325
<b>11 Troubleshooting .....</b>	<b>327</b>
Overview .....	327
Installation .....	327
Communication .....	328
Stability .....	328
Operation .....	328

# Introduction

The LEGEND-MC is a single axis Ethernet motion controller designed for use exclusively with Yaskawa's 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, point-to-point control and communications over the Ethernet connections are standard features. The Ethernet function allows multiple handles or devices to communicate with the controller.

## Part Numbers

		Description	Part Number
SMC3010	a)	Motion Controller with Ethernet Interface	SMC3010
I/O	b)	1.0m 50 Pin I/O Cable	JZSP-CKI01-1 (A)
		2.0m 50 Pin I/O Cable	JZSP-CKI01-2 (A)
		3.0m 50 Pin I/O Cable	JZSP-CKI01-3 (A)
		1.0m 50 Pin I/O Cable (with terminal block)	JUSP-TA50P
Serial	c)	3.0m Port #1 Cable	SMCCBL7
Software	d)	YTermProgrammingSoftware	SMCGUI1

## Startup

### Mounting the LEGEND-MC to the LEGEND Amplifier

1. Insert the lower two mounting notches of the LEGEND-MC into the mounting holes at the bottom of the right side of the LEGEND.
2. Push the LEGEND-MC in the direction indicated by the arrow in the figure below, and insert the upper mounting notches of the LEGEND-MC into the upper mounting holes on the right side of the LEGEND.

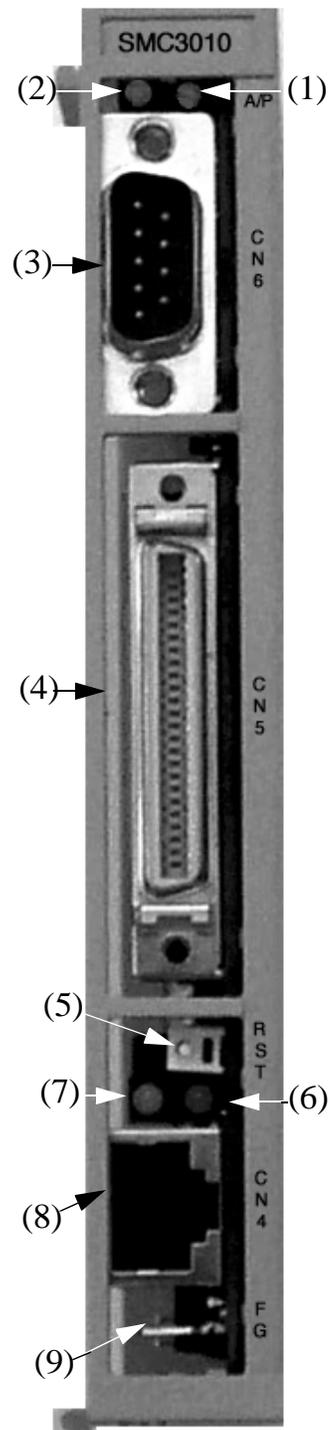
### Mounting Orientation

Mount the LEGEND-MC and LEGEND vertically for proper cooling, as shown below. Allow a minimum spacing of 10mm around the left and right sides and 30mm around the top and bottom of the LEGEND-MC/LEGEND unit.

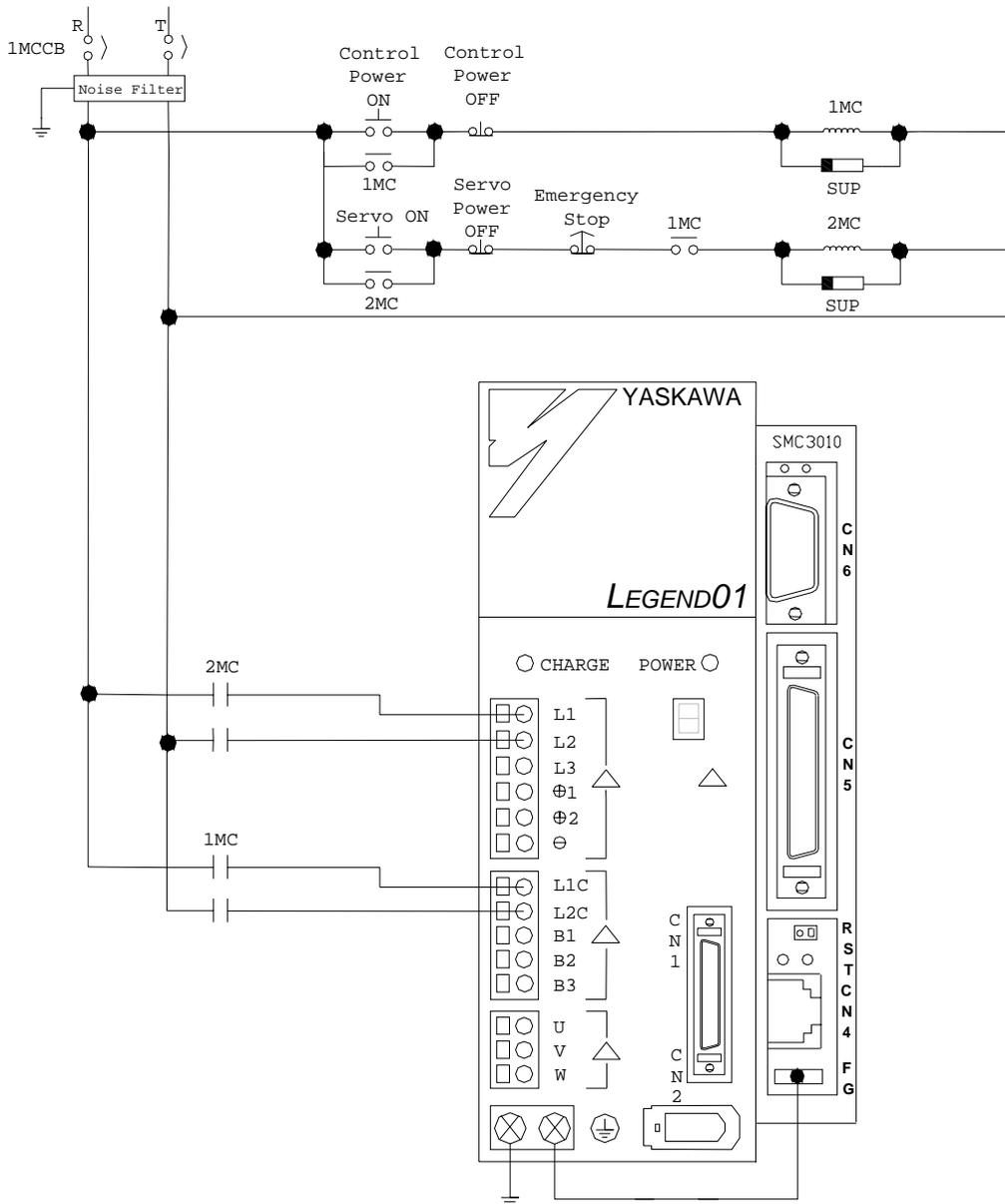


## Front Panel Description

No.	Name	Description
(1)	Power ON	A green LED that indicates +5 VDC power is applied properly from the LEGEND-MC amplifier to the controller.
(2)	Alarm/Error	A red LED that will flash on initially at power up and stay lit for approximately 1-8 seconds. After power up, the LED will illuminate for the following reasons: <ul style="list-style-type: none"> <li>•The axis has a position error greater than the error limit. The error limit is set by using the command ER.</li> <li>•The reset line on the controller is held low or is being affected by noise.</li> <li>•There is a failure in the controller and the processor is resetting itself.</li> <li>•There is a failure in the output IC which drives the error signal.</li> </ul>
(3)	CN6	9 pin male D-Sub serial port connector
(4)	CN5	3M 50 pin high density I/O connector
(5)	RST	Reset switch. 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 tests only for the physical connection, not for an active or enabled link.
(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 RJ485 Connector
(9)	FG	Frame ground spade terminal. Connect to ground terminal on LEGEND Amplifier



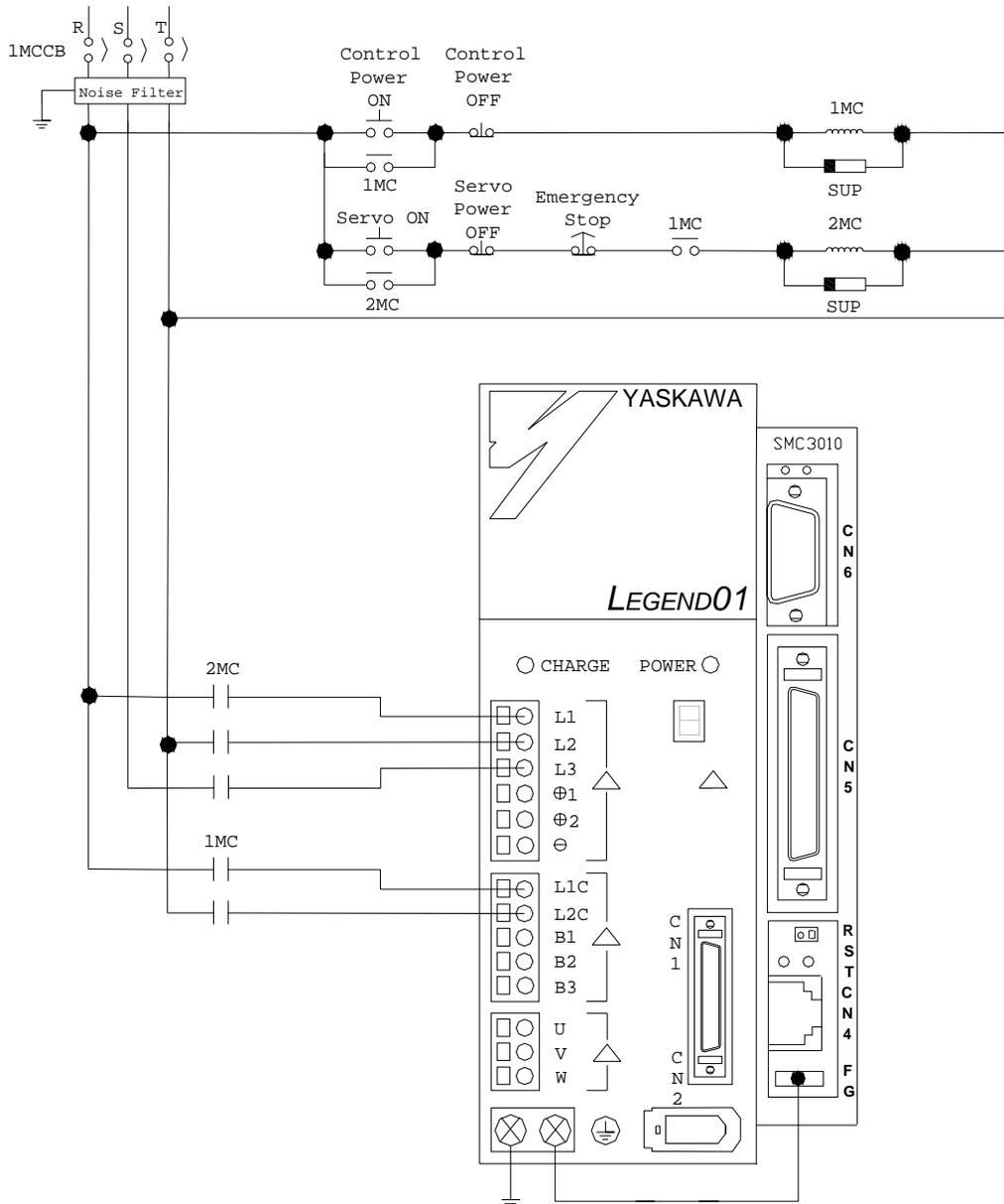
# Power/Connections Wiring - Single Phase



**Notes:** The LEGEND-MC receives its power from the LEGEND amplifier through the side interface connector, however, the digital I/O receives its power from pins 46, 47, 48, and 49 on the I/O connector.

**For maximum noise immunity, connect the FG to a ground terminal on the sub panel or to the ground terminal on the LEGEND.**

# Power/Connections Wiring - Three Phase

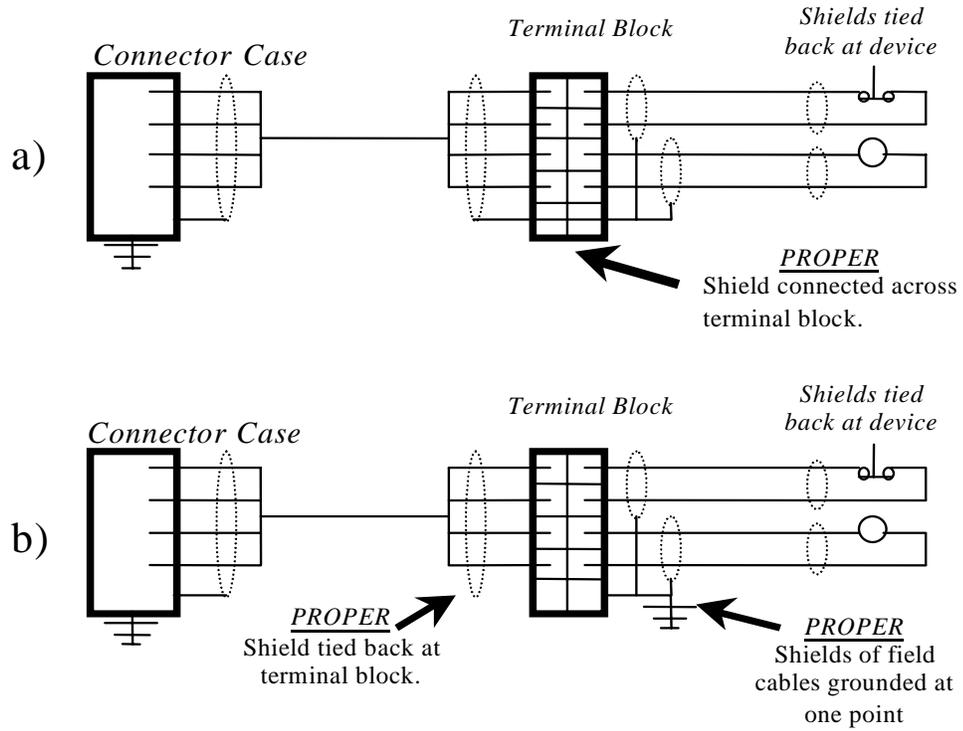


**Notes:** The LEGEND-MC receives its power from the LEGEND amplifier through the side interface connector, however, the digital I/O receives its power from pins 46, 47, 48, and 49 on the I/O connector.

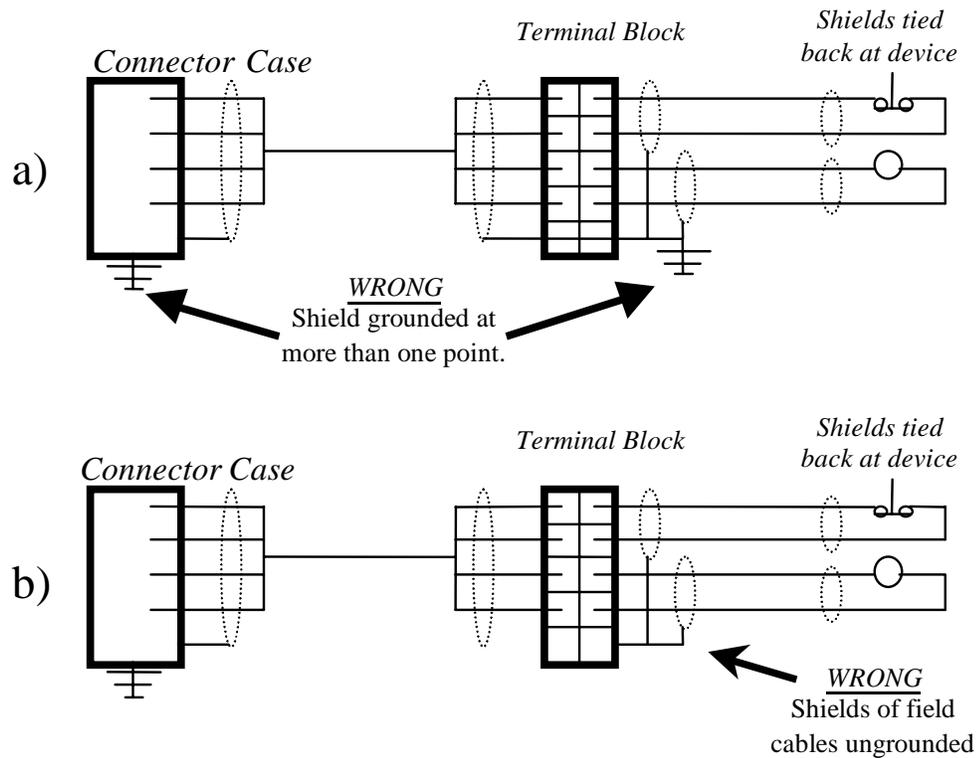
**For maximum noise immunity, connect the FG to a ground terminal on the sub panel or to the ground terminal on the LEGEND.**

# Cable Shielding, Segregation and Noise Immunity

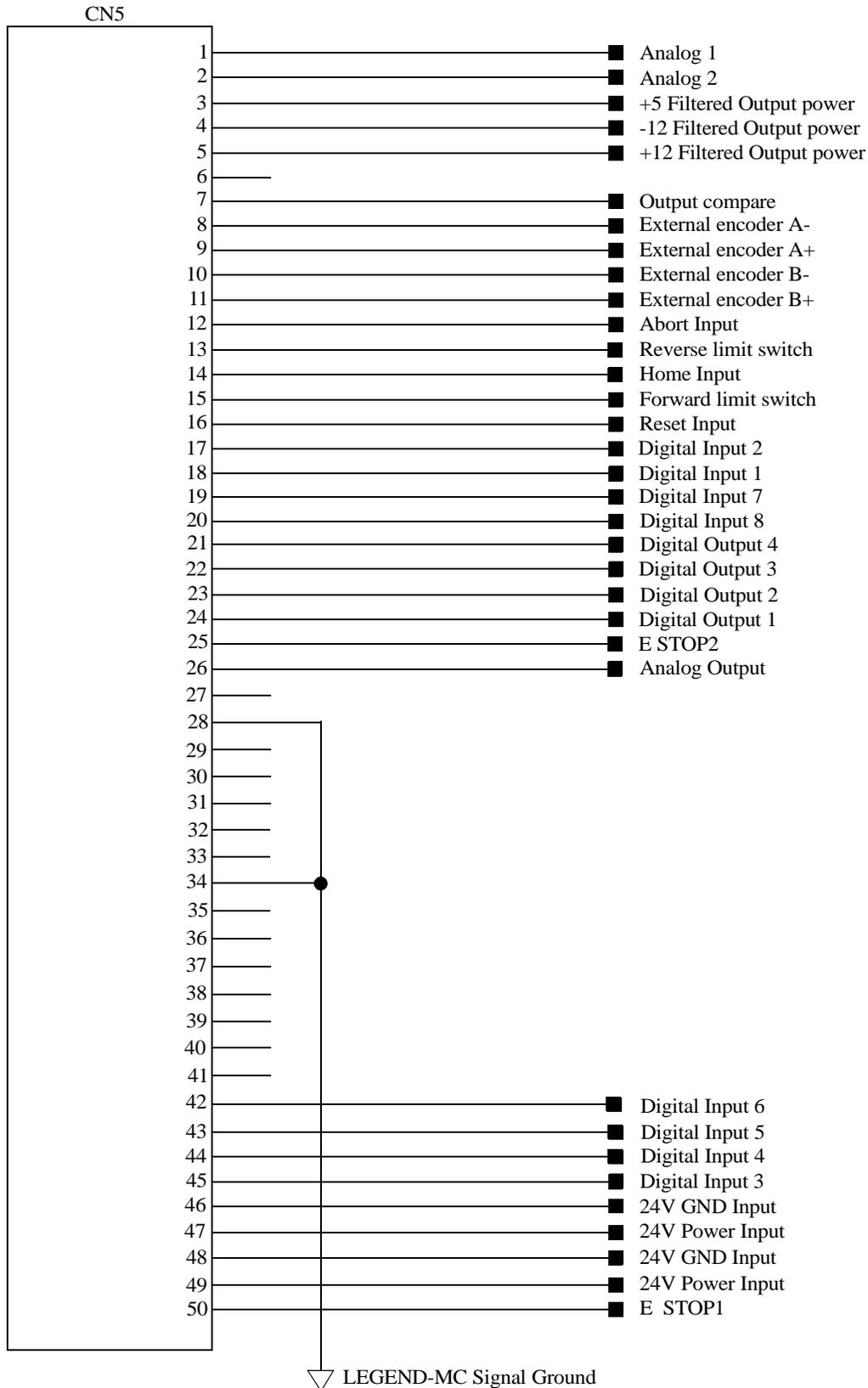
Proper



Wrong



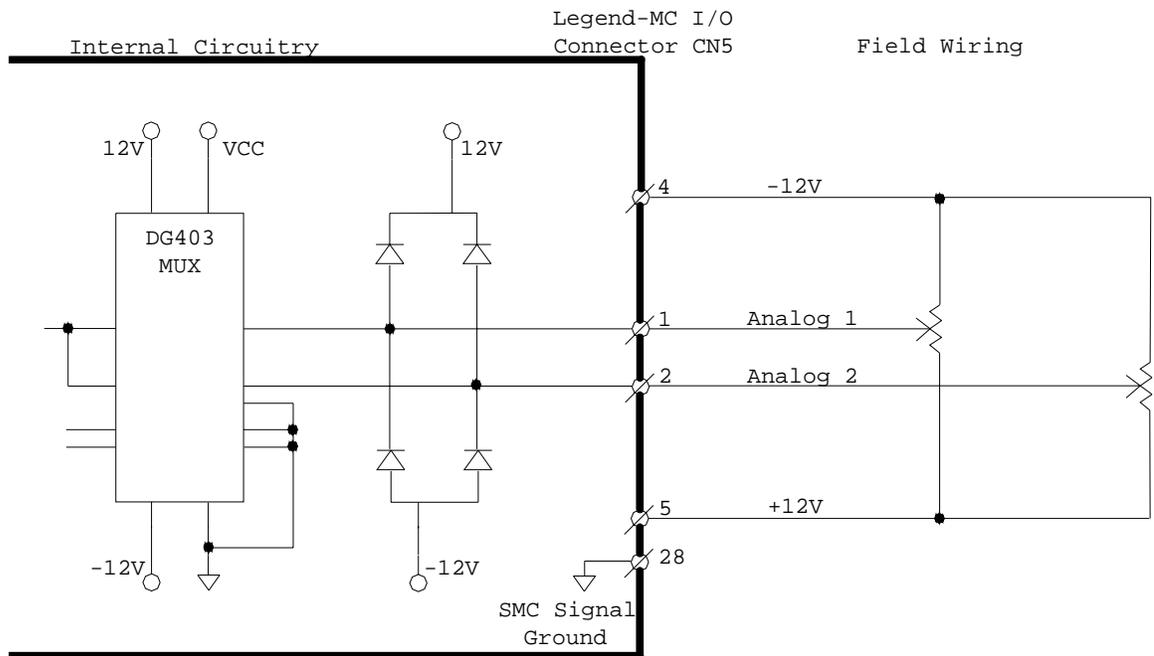
# I/O Connections (50-pin CN5)



# Analog I/O

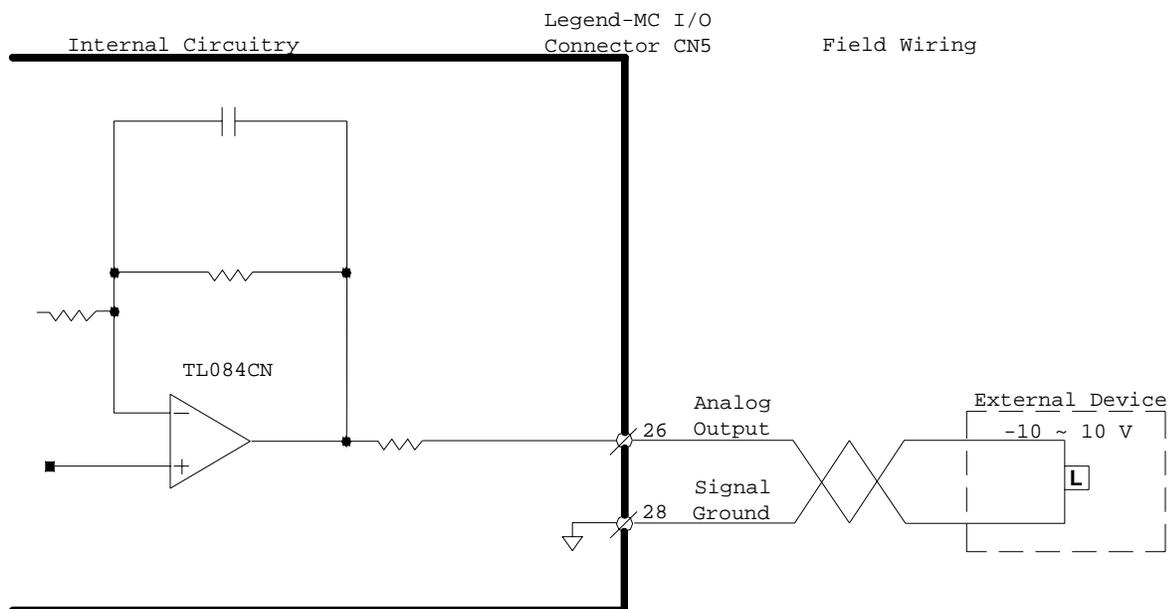
## Analog Input

Item	Specifications
Input Voltage	$\pm 10\text{ V}$
Input Impedance	Approximately $10\text{ k}\Omega$
Resolution	12 bits over a $\pm 10\text{ V}$ range or $4.88\text{ mV}$ per bit



## Analog Output

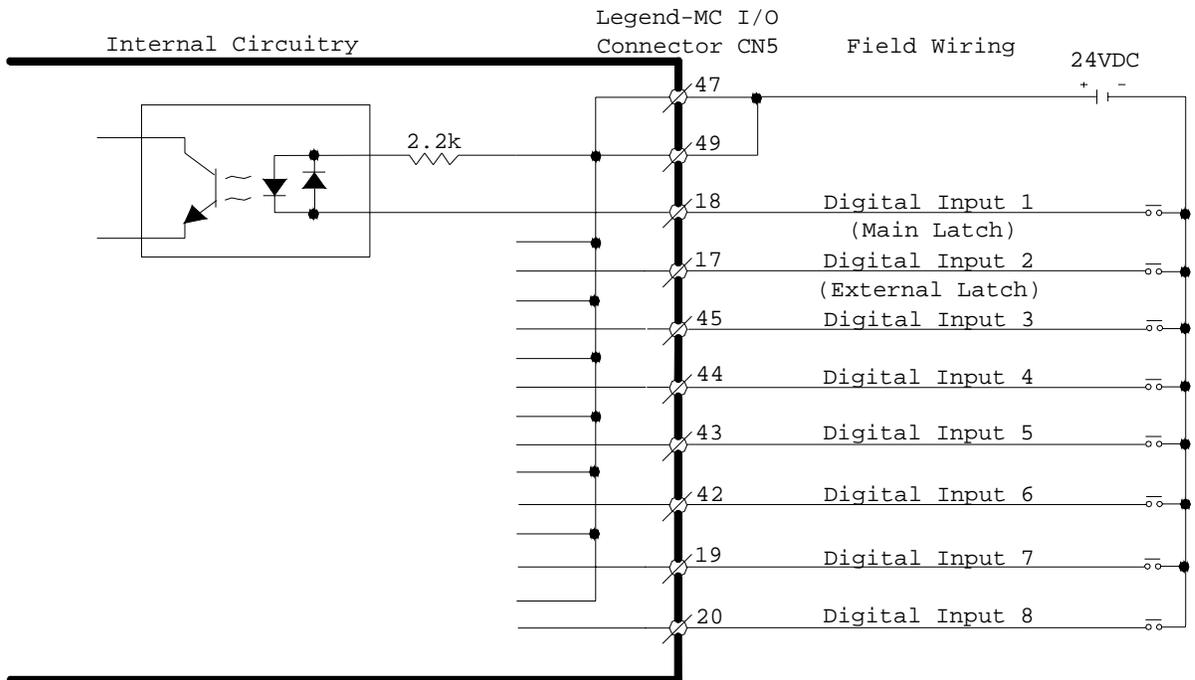
Item	Specifications
D/A Output Resolution	16 bit over a $\pm 10$ V range or $328 \mu\text{V/bit}$
Output short circuit duration	Infinite
Maximum output current	60 mA



# Digital I/O

## Digital Input

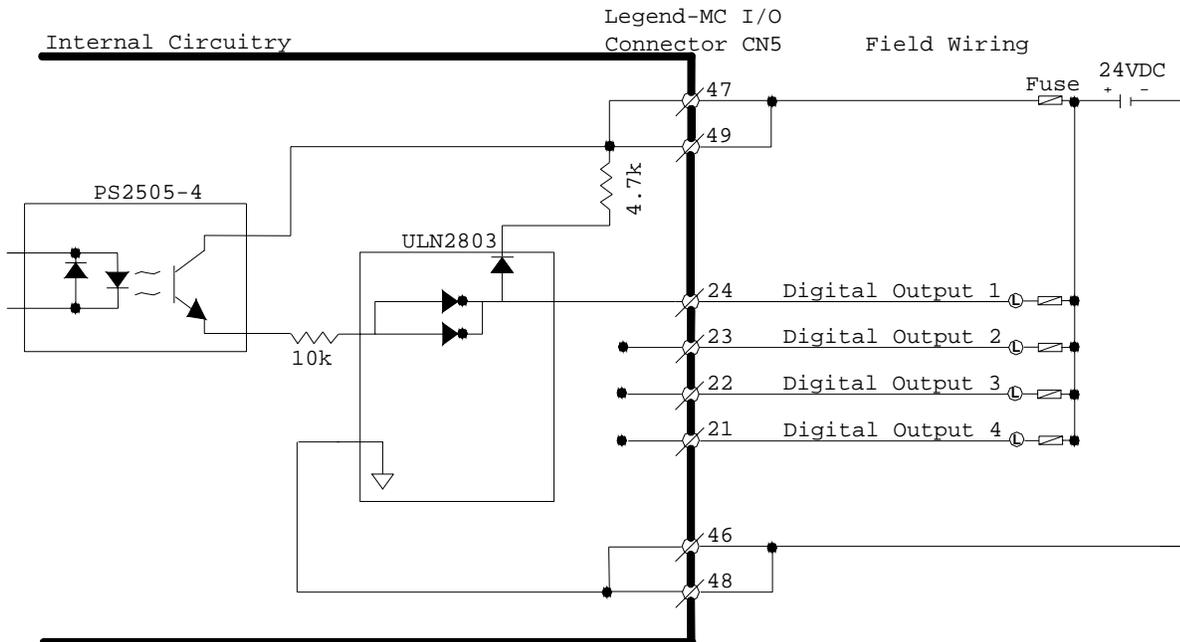
Item	Specifications
Number of Input Points	8
Input Format	Sinking
Isolation	Optical
Voltage	24 VDC $\pm$ 20%
Current Rating (ON)	5.3 mA to activate
Input Impedance	2.2k $\Omega$
Operation Voltage	Logic 0 <5V Logic 1 >15V
OFF Current	0.9 mA or less
Response Time	OFF to ON: <0.5 ms ON to OFF: <1.5 ms
Latch response time	Less than 25 $\mu$ sec
Minimum latch width	9 $\mu$ sec



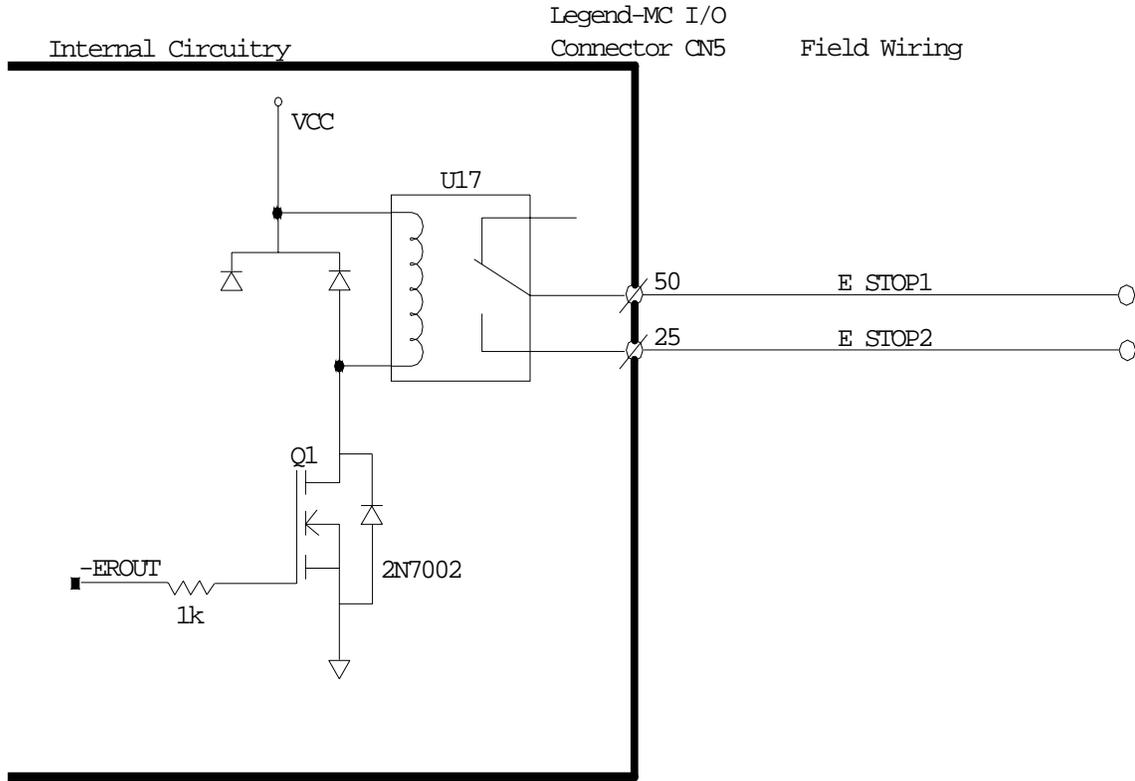
### Digital Output

Item	Specifications
Number of Output Points	4
Output Format	Sinking
Output Classification	Transistor Output
Isolation	Optical
Load Voltage	24 VDC $\pm$ 20%
Load Current	200 mA/Output (600 mA if activated individually)
Response Time	OFF to ON <0.25 ms ON to OFF <0.5 ms
External Common Power	24 VDC $\pm$ 20% 15 mA
Common User Fuse Rating	1A
Individual User Fuse Rating	200 mA recommended

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



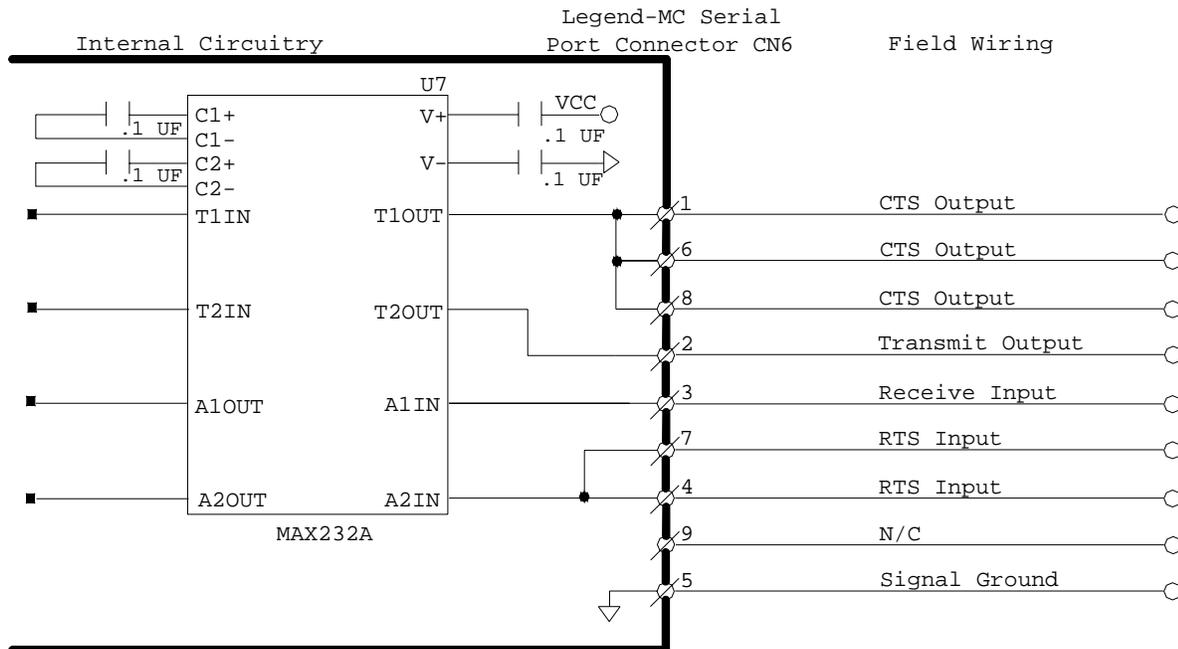
# Emergency Stop Chain



The LEGEND-MC closes the relay contact under normal operating conditions.

# Serial Communication

Item	Specifications
Baud Rate	9600 or 19200 settable by jumper JP1, default is 19200
Data Bits	8
Parity	None
Stop Bits	1

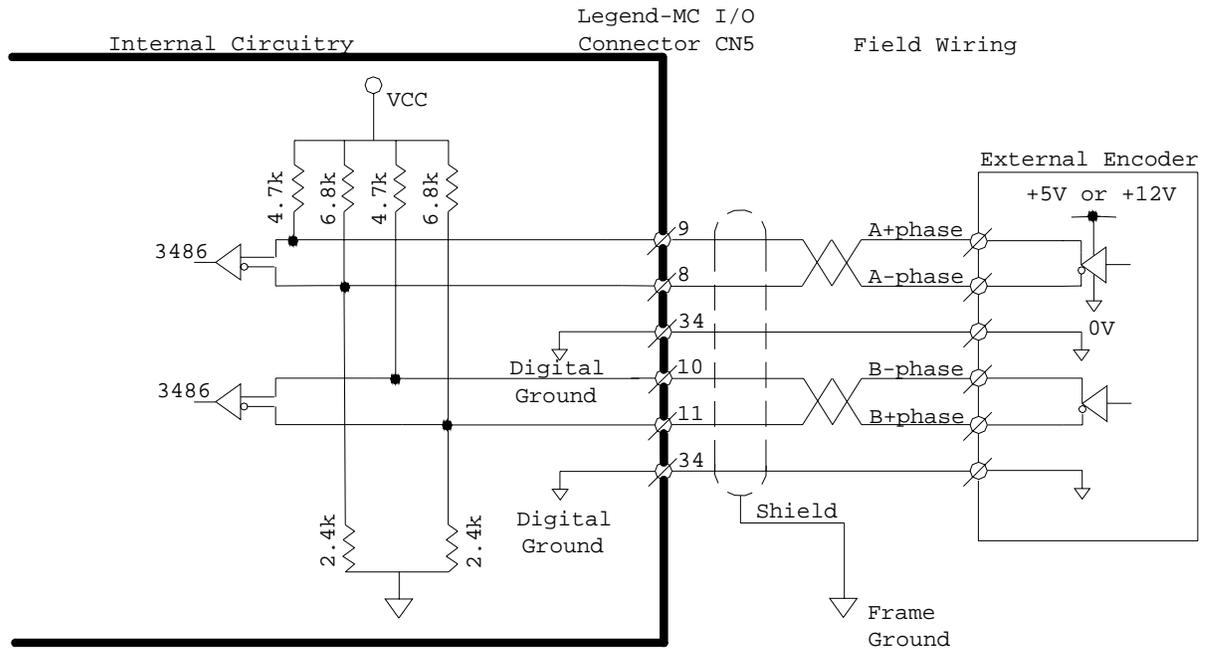


**Note: Hardware handshaking must be used with the LEGEND-MC. If it is impossible to implement hardware handshaking, use a jumper between pins 1 and 4 in the connector.**

**Note: Do not connect pin 5 to a 24V ground.**

## External Encoder Specifications

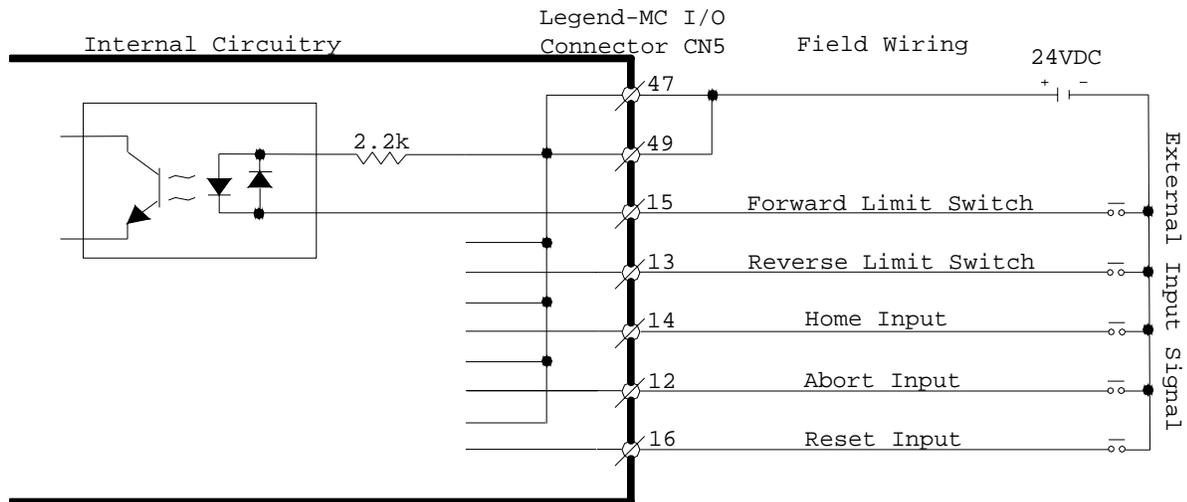
Item	Specifications
Input Format	Quadrature Pulse and Direction
Maximum Frequency	12 MHz



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 complementary input, i.e.; use two 10k resistors, one connected to +12V and the other connected to the LEGEND 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, Abort, Reset
Input Format	Sinking
Isolation	Optical
Voltage	24 VDC $\pm$ 20%
Current Rating (ON)	5.3 mA to activate
Input Impedance	2.2k $\Omega$
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



## Physical Specifications

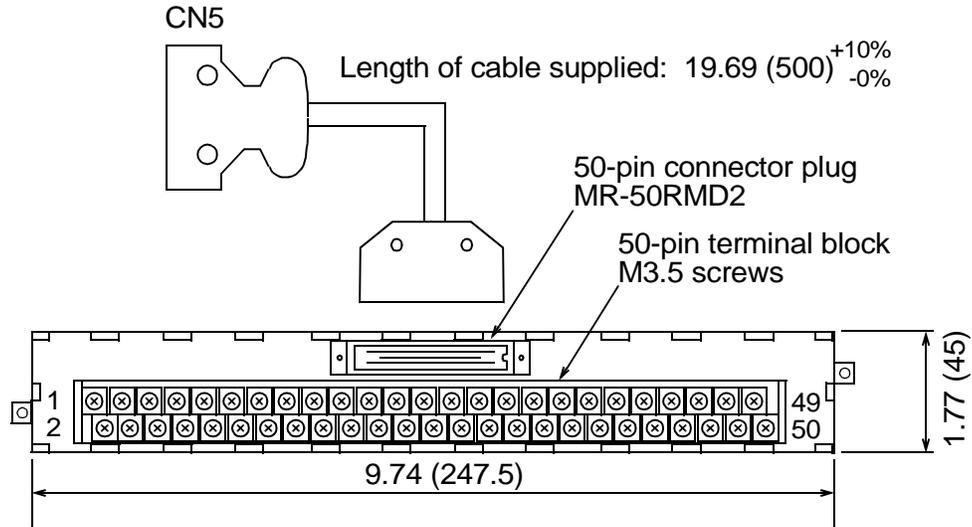
Description	Specifications
Depth:	130mm (5.12 in)
Width:	20 mm (.79 in)
Height:	142 mm (5.6 in)
Weight:	.18 kg (.4 lb.)
Vibration:	9.8 msec <sup>2</sup> (1.0g)
Ambient temperature:	0 ~ 70° C (32 ~ 158° F)
Humidity:	Less than 95%
Noise:	IEC Level 3

## Hardware Specifications

Description	Specifications
CPU:	25 mHz Motorola
Servo update:	1000 μs default, 250 μs minimum
Digital inputs:	(8), +24VDC
Dedicated inputs:	(5), +24VDC
Digital Outputs:	(4), +24VDC
Analog inputs:	(2) +/- 10 V 12 bit resolution
Analog outputs:	(1) +/- 10 V 16 bit resolution
Serial port:	(1) 9600 or 19200 baud
Ethernet:	(1) 10-base-T

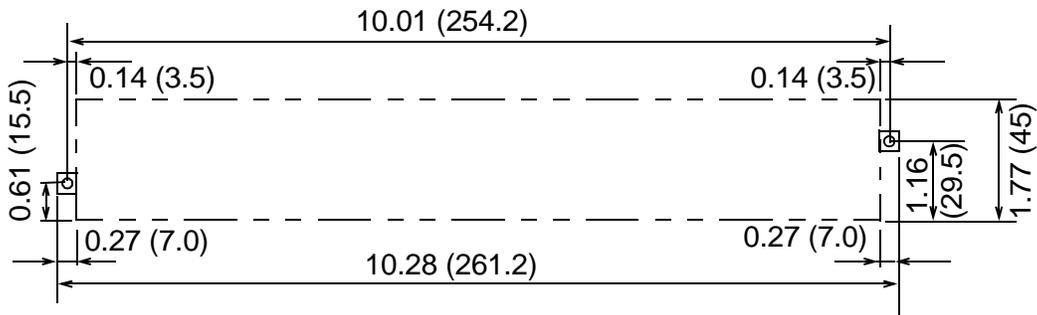
# Cable Diagram and Dimensional Drawings

I/O Cable with Terminal Block JUSP-TA50P

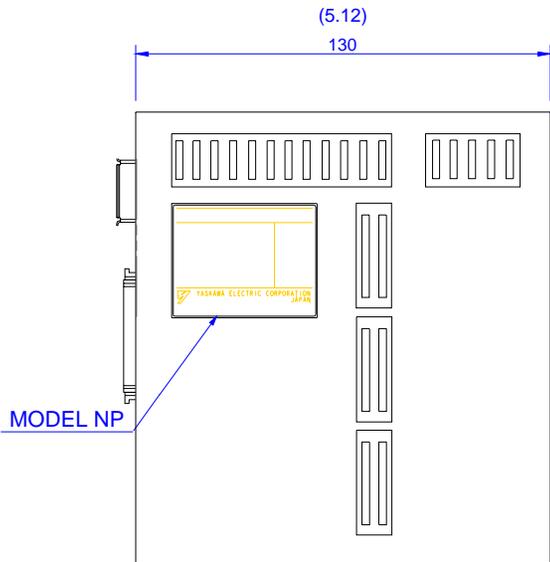
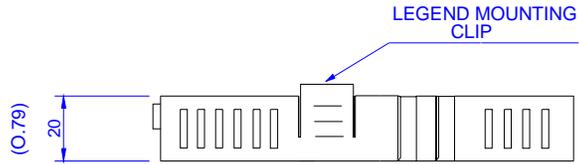
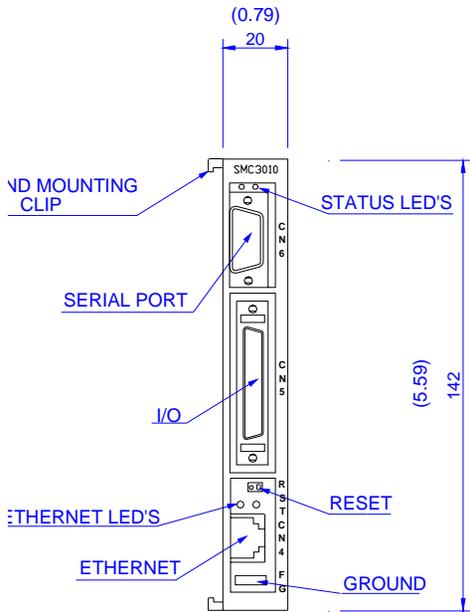


Connector Terminal Block Converter Unit  
JUSP-TA50P\* (cable included)

## Mounting Hole Diagram

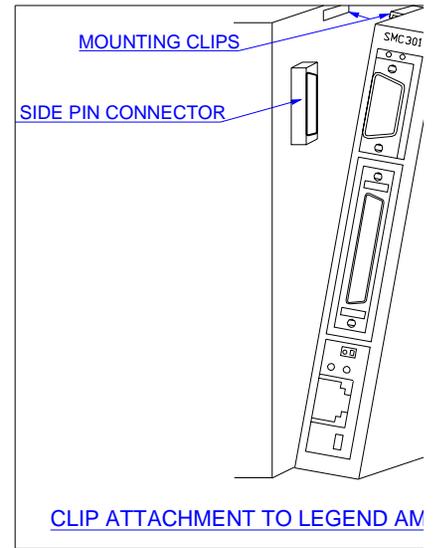


\*Terminal specifications : see I/O connections, page 8



APPROX. MASS: 0.18kg

DIMENSIONS: MM (IN)

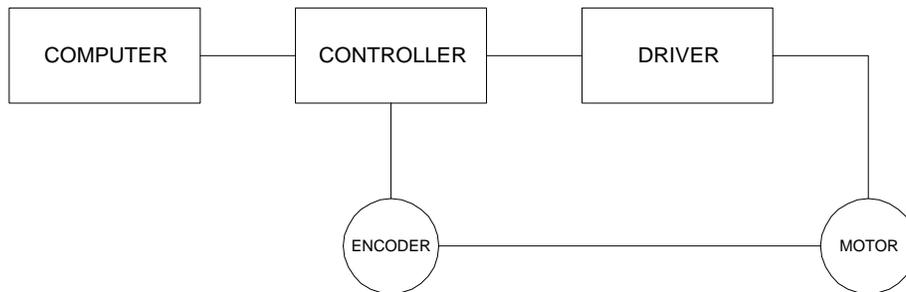


**NOTES:**

## 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:

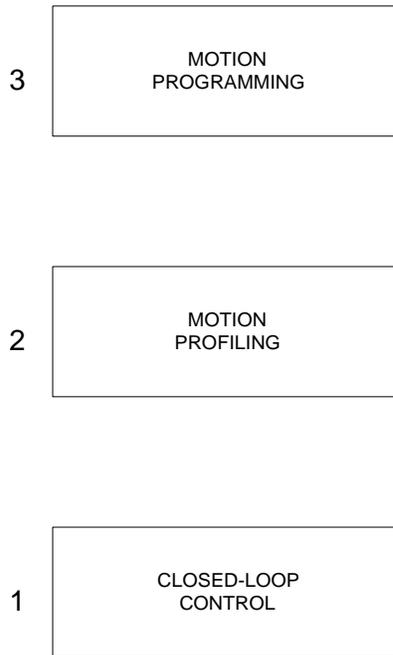
1. Closing the Loop
2. Motion Profiling
3. Motion Programming

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.

# Level



*Levels of Control Functions*

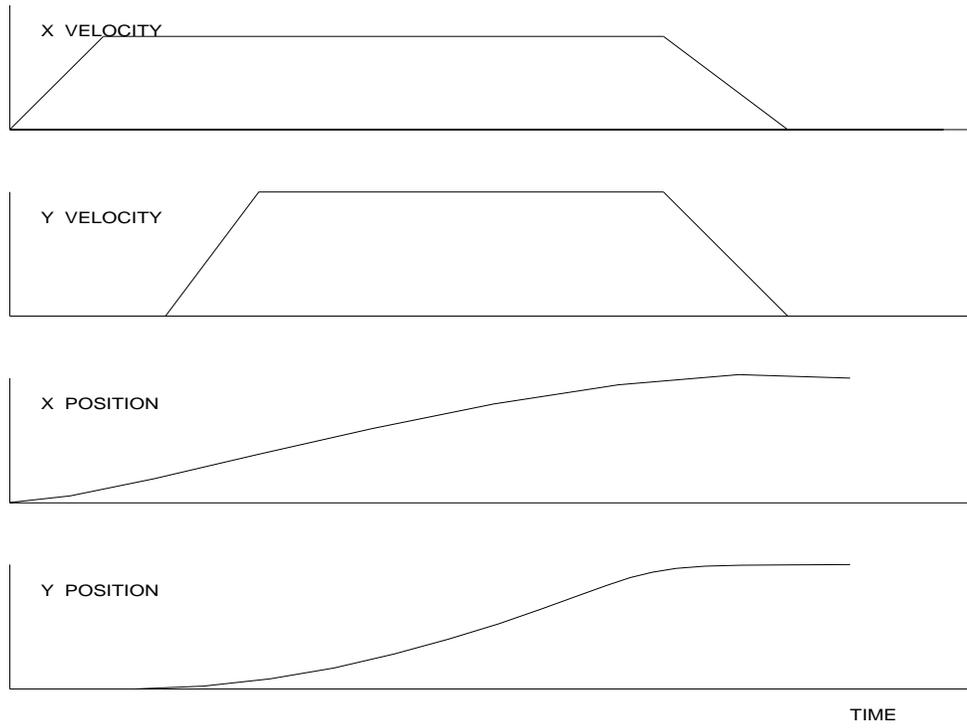
The three levels of control may be viewed as different levels of management. The top manager, the motion program, may specify the following instruction, for example.

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.

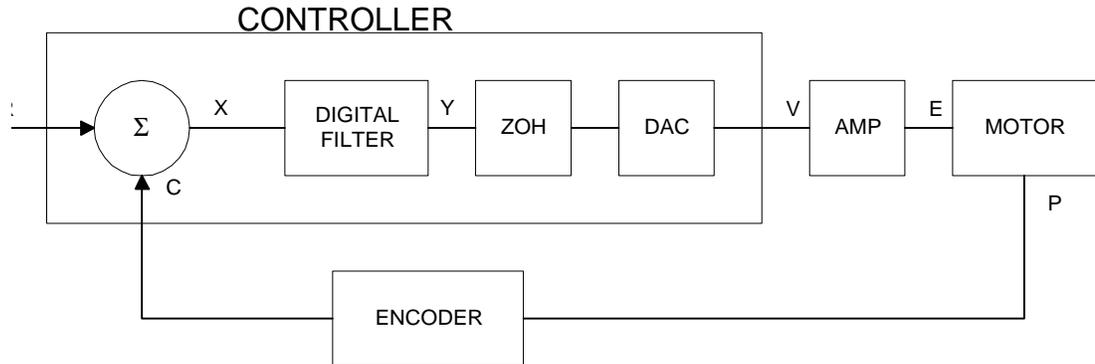
The motor position, whether rotary or linear is measured by a sensor. The resulting signal, called position feedback, is returned to the controller for closing the loop.

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

## System Modeling

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 may be configured in two modes:

1. Current Drive
2. Velocity Loop

The operation and modeling in the two modes is as follows:

#### Current Drive

The current drive generates a current  $I$ , which is proportional to the input voltage,  $V$ , with a gain of  $K_a$ , 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  $K_a = 2 \text{ A/V}$  with the motor described by the previous example will have the transfer function:

$$P/V = 1000/s^2 \quad [\text{rad/V}]$$

## Velocity Loop

The motor driver system may include a velocity loop where the motor velocity is sensed by a tachometer and is fed back to the amplifier. Such a system is shown in the following illustration. Note that the transfer function between the input voltage  $V$  and the velocity  $\omega$  is:

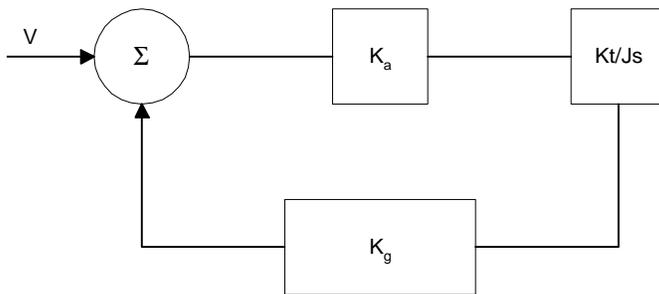
$$\omega / V = [K_a K_t / Js] / [1 + K_a K_t K_g / Js] = 1 / [K_g (sT_1 + 1)]$$

where the velocity time constant,  $T_1$ , equals:

$$T_1 = J / K_a K_t K_g$$

This leads to the transfer function:

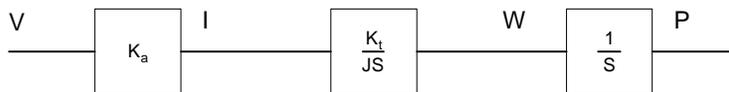
$$P/V = 1 / [K_g s (sT_1 + 1)]$$



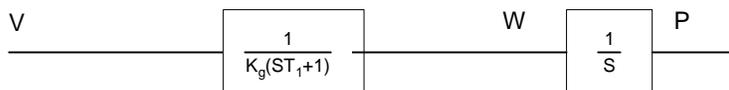
*Elements of velocity loops*

The functions derived above are shown in the following block diagram :

### CURRENT SOURCE



### VELOCITY LOOP



*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:

$$K_f = 4N/2\pi \quad [\text{count/rad}]$$

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

$$K_f = 638$$

## DAC

The DAC or D-to-A converter converts a 14-bit number to an analog voltage. The input range of the numbers is 16384 and the output voltage range is +/-10V or 20V. Therefore, the effective gain of the DAC is:

$$K = 20/16384 = 0.0012 \quad [\text{V/count}]$$

## 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  $G(s)$ .

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

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

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

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

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

$$KD = 36$$

$$KI = 2$$

$$T = 0.001 \text{ s}$$

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 LEGEND-MC controller and the following parameters:

$K_t = 0.1$	Nm/A	Torque constant
$J = 2.10^{-4}$	kg.m <sup>2</sup>	System moment of inertia
$R = 2$	$\Omega$	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:

$$M(s) = P/I = K_t/Js^2 = 500/s^2 \text{ [rad/A]}$$

Amp:

$$K_a = 4 \text{ [Amp/V]}$$

DAC:

$$K_d = 0.0012 \text{ [V/count]}$$

Encoder:

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

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:

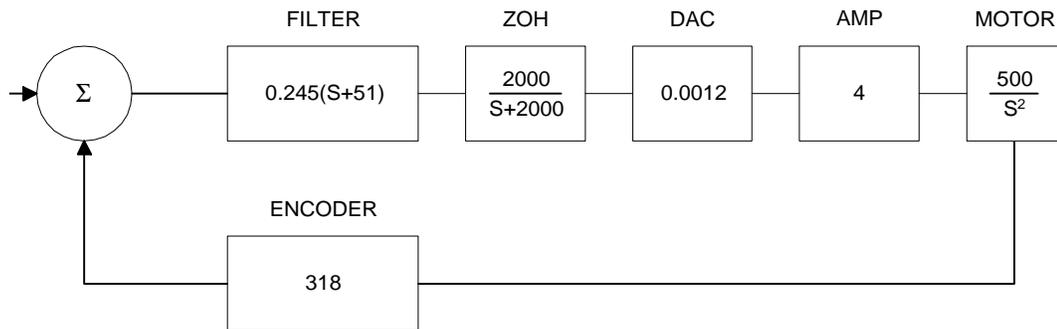
$$P = 12.5$$

$$D = 0.245$$

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:

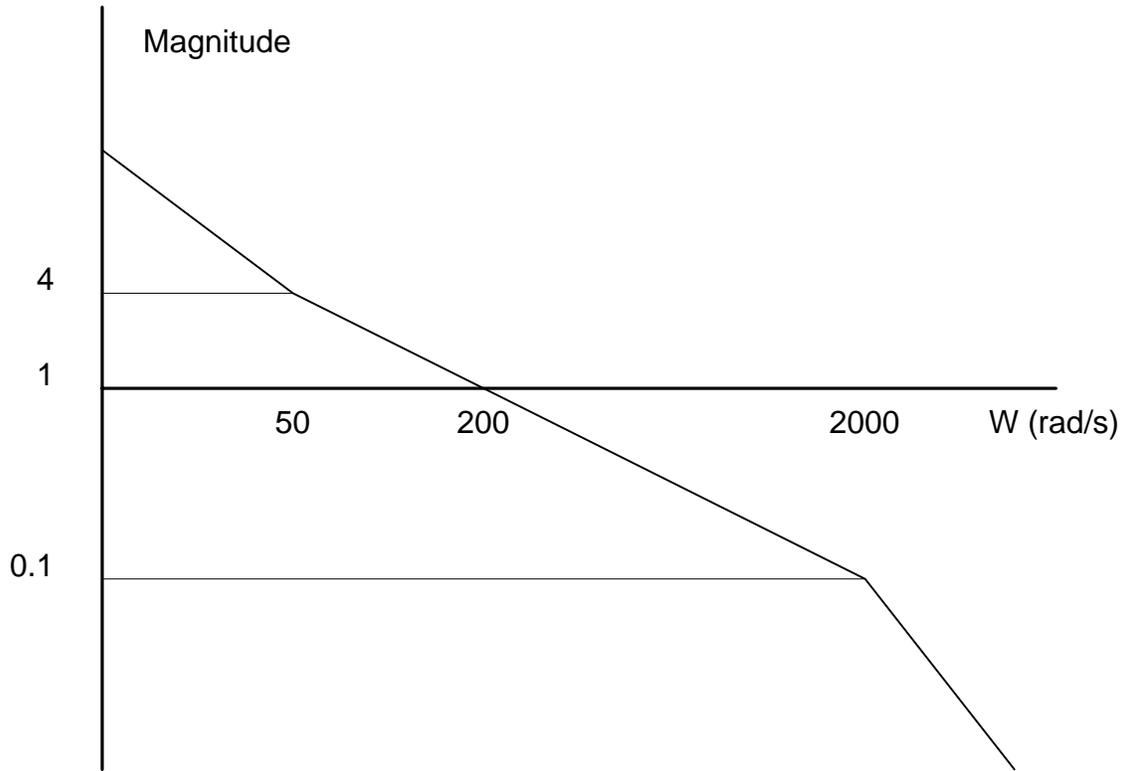


*Mathematical model of the control system*

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,  $\omega_c$  at which  $A(j \omega_c)$  equals one. This can be done by the Bode plot of  $A(j \omega_c)$ , 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:

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

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

$$\alpha = 76^\circ - 180^\circ - 6^\circ = -110^\circ$$

Finally, the phase margin, PM, equals:

$$\text{PM} = 180^\circ + \alpha = 70^\circ$$

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 preprogrammed in the LEGEND-MC 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,  $\omega_c$ , 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
$J = 2 \cdot 10^{-4}$	kg.m <sup>2</sup>	System moment of inertia
$R = 2$	$\Omega$	Motor resistance
$K_a = 2$	Amp/Volt	Current amplifier gain
$N = 1000$	Counts/rev	Encoder line density

The DAC of the LEGEND-MC outputs +/-10V for a 14-bit command of +/-8192 counts.

The design objective is to select the filter parameters in order to close a position loop with a crossover frequency of  $\omega_c = 500$  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:

$$K_a = 2 \quad [\text{Amp/V}]$$

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,  $L(s)$ :

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

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  $\omega_c = 500$ :

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

This function has a magnitude of:

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

and a phase:

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

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

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

$$\text{Arg}[A(j500)] = -135^\circ$$

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:

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

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:

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

The solution of these equations leads to:

$$P = 40 \cos 59^\circ = 20.6$$

$$500D = 40 \sin 59^\circ = 34.3$$

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  $T=1\text{ms}$ , the parameters of the digital filter are:

$$K_P = 20.6$$

$$K_D = 68.6$$

The LEGEND-MC can be programmed with the instruction:

$$K_P \ 20.6$$

$$K_D \ 68.6$$

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	$D(z) = K_P + K_D(1-z^{-1}) + K_I/8(1-z^{-1})$
$K_P, K_D, K_I$	$K = K_P + K_D$ $A = K_D/(K_P+K_D)$ $C = K_I/8$
Digital	$D(z) = G_N(z-ZR)/z + K_I z/8(z-1)$
$G_N, ZR, K_I$	$K = G_N$ $A = ZR$ $C = K_I/8$
Continuous	$G(s) = P + Ds + I/s$
PID, T	$P = K(1-A) = K_P$ $D = K.A.T = T.K_D$ $I = C/T = K_I/8T$

**NOTES:**

## 3 Communications

### Introduction

The LEGEND-MC has one RS232 port and one Ethernet port. The RS-232 port is the data set. 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 LEGEND-MC 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 LEGEND-MC decodes each ASCII character (one byte) one at a time. It takes approximately .5 msec for the controller to decode each command.

After the instruction is decoded, the LEGEND-MC 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 as Tell Position (TP), the LEGEND-MC 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 LEGEND-MC response with the data sent. The echo is enabled by sending the command EO 1 to the controller.

### RS232 Port

The LEGEND-MC 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 DATATERM***

1 CTS – output	6 CTS – output
2 Transmit Data - output	7 RTS – input
3 Receive Data - input	8 CTS – output
4 RTS – input	9 No connection (Can connect to +5V, 30mA)
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 your PC for 8-bit data, one start-bit, one stop-bit, full duplex and no parity. The baud rate for the RS232 communication can be chosen by selecting the proper jumper configuration on the LEGEND-MC according to the table below.

JP1-LOCATION "96" (JUMPER ATTACHED)	9600
JP1-LOCATION "96" (JUMPER UNATTACHED)	19200

## 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 LEGEND-MC is not ready to receive additional characters. The RTS line will inhibit the LEGEND-MC from sending additional characters.

**Note: The RTS line goes high for inhibit. This handshake procedure ensures proper communication especially at higher baud rates.**

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

# 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 LEGEND-MC 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 acknowledgment 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.**

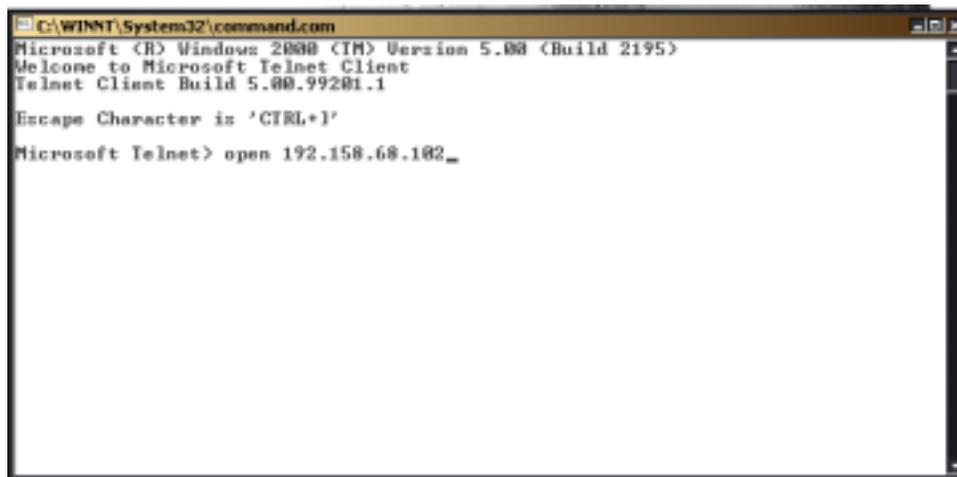
## Addressing

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. No other device has the same Ethernet address. The LEGEND-MC 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.

The first method is to use the BOOT-P utility via the Ethernet connection (the LEGEND-MC 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.**



```
C:\WINNT\System32\command.com
Microsoft (R) Windows 2000 (TM) Version 5.00 (Build 2195)
Welcome to Microsoft Telnet Client
Telnet Client Build 5.00.99281.1
Escape Character is 'CTRL+I'
Microsoft Telnet> open 192.158.68.182
```

The second method for setting an IP address is to send the IA command through the LEGEND-MC 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.**

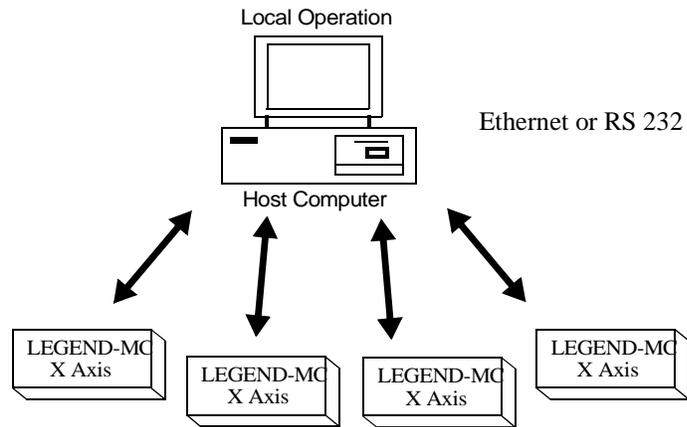
The third level of Ethernet addressing is the UDP or TCP port number. The Yaskawa controller does not require a specific port number. The port number is established by the master each time it connects to the controller.

## Ethernet Handles

An Ethernet handle is a communication resource within a device. The LEGEND-MC can have a maximum of 8 Ethernet handles open at any time. When using TCP/IP, each connection to a device; i.e., the host computer, requires an individual Ethernet handle. In UDP/IP, one handle may be used for all the masters, but each slave uses one. Pings and ARPS do not occupy handles. If all 8 handles are in use and a 9<sup>th</sup> master tries to connect, it will be sent a "reset packet" that generates the appropriate error in its native application.

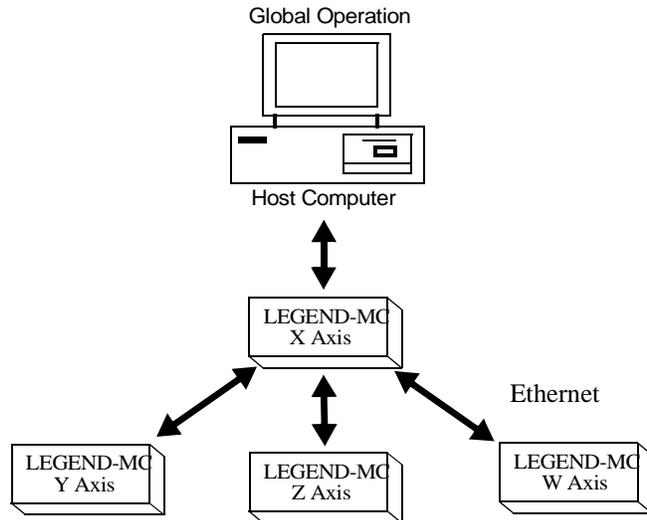
## Global vs Local Operation

Each LEGEND-MC controls one axis of motion, referred to as A or X. The host computer can communicate directly with any LEGEND-MC via an Ethernet or RS-232 connection. When the host computer is directly communicating with any LEGEND-MC, all commands refer to the first axis as A or X. Direct communication with the LEGEND-MC is known as "local operation". The concept of Local and Global Operation also applies to application programming.



The LEGEND-MC supports Yaskawa's control system. This allows up to four LEGEND-MC controllers to be connected together as a single virtual axis controller. In this system, one of the controllers is designated as the master. The master can receive commands from the host computer that apply to all of the axes in the system.

Here is a simple way to view Local and Global Operation; when the host communicates with a slave controller, it considers the slave as a 1 axis controller. When the host communicates with a master, it considers the master as a multi-axis controller. Similarly, an application program residing in a slave controller deals only with 1 motor as A or X. An application program in a master deals with all motors referenced A through H.



The controllers may operate under both Local and Global Mode. In general, operating in Global Mode simplifies controlling the entire system. However, Local Mode operation is necessary in some situations; using Local Mode for setup and testing is useful since this isolates the controller. Specific modes of motion require operation in Local Mode. Also, each controller can have a program including the slave controllers. When a slave controller has a program, this program would always operate in Local Mode.

## Configuring Operation for Distributed Control

Each LEGEND-MC must be assigned an IP address. This can be done with the BOOT-P procedure or the IA command can be used to assign the IP address through the serial port. Once the IP address has been assigned, a BN command should be issued to save this value in the controller's non-volatile memory.

Upon power-up or reset, the master LEGEND-MC will establish each slave connection. The following steps must be taken while connected to the master LEGEND-MC:

1. Using the IH command, open two handles for each slave. Each slave controller must have 2 open handles, one for commands from the master, the other for data returned from the slave. The second internet handle for each slave controller must contain a specific port value. The value must be an even number greater than 502. The command for opening the communication handle is:

IHh=ip0,ip1,ip2,ip3<p>2 h is the handle. ip is the slave IP address. <p> specifies port number. >2 specifies TCP/IP.

2. Set the total number of axes in the system with the NA command. For example, assume there are two LEGEND-MC slaves, therefore there will be three axes and the command would be NA3.

3. Connect each slave handle to the master. This is accomplished with the CH command. The format of this command is:

CHa=h1,h2 where a is the first axis designator of the slave controller, h1 is the handle for commands and h2 is the handle for slave status.

4. For the master controller to make decisions based on the status of the slave controllers, it is necessary for the slaves to generate data records giving their current status. The record is sent at a rate set by the QW command. The QW command must be executed by the master before the slave can issue a record under any method. The format of the command is:

QWh=n where h is the handle. n is a number between 4 and 16000.

n sets the number of samples (msec with default TM1000).

n equal to 0 disables the mode.

The data contained in the record is as follows:

- (RP) reference position
- (TP) encoder position
- (TE) position error
- (TV) velocity
- (TT) torque
- (TS) limit and home switches
- (TS) axis status (in motion, motor of, at speed, stopcode)
- (TI) uncommitted inputs
- (OP) uncommitted outputs
- (ZA) user defined variables

## Operation of Distributed Control

For most commands it is unnecessary to be conscious of whether an axis is local or remote. For example, to set the KP value for the X and Y axes, the command for the master would be:

```
KP 10,,20
```

Similarly, the interrogation commands can also be issued. For example, the position error for all axes would be TE. The position operand for the F axis would be \_TPF.

Some commands are inherently sent to all controllers. These include commands such as AB (Abort), CN and TM.

Certain commands need to be launched specifically. 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 slave, it is possible for the master to determine the status of the command. The response \_IHh4 will return the number 1 to 4. One means waiting for the acknowledgment from the slave. Two means a colon (command accepted) has been received. Three means a question mark (command rejected) has been received. Four means the command timed out.

If a command generates 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.

## Accessing the I/O of the slaves

The I/O of the slaves are settable and readable from the master. The bit numbers are adjusted by the handle number of the data record. Each handle adds 100 to the bit number. Handle A is 100 and Handle F is 600.

### **Example**

Bit 2 on the slave using handle E for the data record would be 502. The arguments for SB, CB, and OB use this format as does the @IN[ ] function.

For byte and word-wide I/O, use the SA command such as: SAC="OP6" to set the output port of handle C. SAC="TI" will return the input port on handle C and the operand, \_SAC0 will contain the response from the TI command.

## 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 section.

### **Example**

Assume a system with one master and 2 slave LEGEND-MC controllers:

<b>Instruction</b>	<b>Interpretation</b>
#SETUP	Begin Program
IHF=160,50,10,1>2	Set handle F (for commands) to slave 1's IP
IHE=160,50,10,1<510>2	Open handle E for 1's data record
IHD=160,50,10,2>2	Set the handle D (for commands) to slave 2's IP
IHC=160,50,10,2<512>2	Open handle E for 2's data record
NA3	3 axis total
CHY=F,E	Axis Y assigned to slave 1
CHZ=D,C	Axis Z assigned to slave 2
QWE=20	Handle E sends record every 20 msec
QWC=20	
EN	

## 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 LEGEND-MC can use a specific slave address or default to the handle number.

Modbus protocol has commands called function codes. The LEGEND-MC 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 LEGEND-MC provides three levels of Modbus communication. 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

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

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:

$$\text{I/O Number} = (\text{HandleNum} * 1000) + ((\text{Module} - 1) * 4) + (\text{BitNum})$$

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:

$$\text{I/O Number} = (\text{SlaveAddress} * 1000) + (\text{HandleNum} * 1000) + ((\text{Module} - 1) * 4) + (\text{Bitnum} - 1)$$

Which devices receive what information from the controller depends on a number of things. If a device queries the controller, it will receive the response unless it explicitly tells the controller to send it to another device. If the command that generates a response is part of a downloaded program, the response will route to whichever port is specified as the default (unless explicitly told to go to another port). To designate a specific destination for the information, add {Eh} to the end of the command. (Ex. MG{EC}"Hello" will send the message "Hello" to handle #3. TP.,?{EF} will send the z axis position to handle #6.)

## Communicating with Multiple Devices

The LEGEND-MC is capable of supporting multiple 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.

An Ethernet handle is a communication resource within a device. The LEGEND-MC can have a maximum of 8 Ethernet handles open at any time. When using TCP/IP, each master or slave uses an individual Ethernet handle. In UDP/IP, one handle may be used for all the masters, but each slave uses one.

(Pings and ARP's do not occupy handles.) If all 8 handles are in use and a 9<sup>th</sup> master tries to connect, it will be sent a "reset packet" that generates the appropriate error in its native application.

## Multicasting

A multicast is only used in UDP/IP and is similar to a broadcast (everyone on the network gets the information) but specific to a group. As such, all devices within a specified group will receive information sent in a multicast. The many multicast groups on a network are differentiated by their multicast IP address. To communicate with all devices in a specific multicast group, information can be sent to the multicast IP address rather than to each device IP address. All LEGEND-MC controllers belong to a default multicast address of 239.255.19.56. The controller's multicast IP address can be changed by using the IA> u command.

## Using Third Party Software

Yaskawa supports ARP, BOOT-P, and Ping, which are utilities for establishing Ethernet connections.

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 LEGEND-MC 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

**AB (ABORT)**  
**@ABS (ABSOLUTE VALUE)**  
**AC (ACCELERATION)**  
**@ACOS (ARC COSINE)**  
**AD (AFTER DISTANCE)**  
**AF (ANALOG FEEDBACK)**  
**AI (AFTER INPUT)**  
**AL (ARM LATCH)**  
**AM (AFTER MOTION)**  
**@AN (READ ANALOG)**  
**AO (ANALOG OUTPUT)**  
**AP (AFTER ABSOLUTE POSITION)**  
**AR (AFTER RELATIVE DISTANCE)**  
**AS (AT SPEED)**  
**@ASIN (ARC SINE)**  
**AT (AT TIME)**  
**@ATAN (ARC TANGENT)**  
**BG (BEGIN MOTION)**  
**BL (REVERSE SOFTWARE LIMIT)**  
**BN (BURN)**  
**BP (BURN PROGRAM)**  
**BV (BURN VARIABLES)**  
**CB (CLEAR BIT)**  
**CD (CONTOUR DATA)**  
**CE (CONFIGURE ENCODER)**  
**CF (CONFIGURE MESSAGES)**  
**CH (CONNECT HANDLE)**  
**CI (COMMUNICATION INTERRUPT)**  
**CM (CONTOUR MODE)**  
**CN (CONFIGURE LIMIT SWITCHES)**  
**@COM (2'S COMPLEMENT)**  
**@COS (COSINE)**  
**CS (CLEAR SEQUENCE)**  
**CW (COPYRIGHT INFORMATION / DATA ADJUSTMENT BIT ON/OFF)**  
**DA (DE-ALLOCATE THE VARIABLES & ARRAYS)**  
**DC (DECELERATION)**  
**DE (DUAL (AUXILIARY) ENCODER POSITION)**  
**DL (DOWNLOAD)**  
**DM (DIMENSION)**  
**DP (DEFINE POSITION)**  
**DT (DELTA TIME)**

**DV (DUAL VELOCITY (DUAL LOOP))**  
**EA (ECAM MASTER AXIS)**  
**EB (ENABLE ECAM MODE)**  
**EC (ECAM COUNTER)**  
**ED (EDIT)**  
**EG (ECAM ENGAGE)**  
**ELSE (ELSE FUNCTION FOR USE WITH IF CONDITIONAL STATEMENT)**  
**EM (ECAM CYCLE)**  
**EN (END)**  
**ENDIF (END OF IF CONDITIONAL STATEMENT)**  
**EO (ECHO)**  
**EP (CAM TABLE INTERVALS AND STARTING POINT)**  
**EQ (ECAM QUIT (DISENGAGE))**  
**ER (ERROR LIMIT)**  
**ET (ELECTRIC CAM TABLE)**  
**FA (ACCELERATION FEED FORWARD)**  
**FE (FIND EDGE)**  
**FI (FIND INDEX)**  
**FL (FORWARD SOFTWARE LIMIT)**  
**@FRAC (FRACTION)**  
**FV (VELOCITY FEED FORWARD)**  
**GA (MASTER AXIS FOR GEARING)**  
**GR (GEAR RATIO)**  
**HM (HOME)**  
**HX (HALT EXECUTION)**  
**IA (IP ADDRESS)**  
**IF (IF CONDITIONAL STATEMENT)**  
**IH (OPEN INTERNET HANDLE)**  
**II (INPUT INTERRUPT)**  
**IL (INTEGRATOR LIMIT)**  
**IN (INPUT VARIABLE)**  
**@IN (STATUS OF DIGITAL INPUT)**  
**@INT (INTEGER)**  
**IP (INCREMENT POSITION)**  
**IT (INDEPENDENT TIME CONSTANT - SMOOTHING FUNCTION)**  
**JG (JOG)**  
**JP (JUMP TO PROGRAM LOCATION)**  
**JS (JUMP TO SUBROUTINE)**  
**KD (DERIVATIVE CONSTANT)**  
**KI (INTEGRATOR)**  
**KP (PROPORTIONAL CONSTANT)**  
**LA (LIST ARRAYS)**  
**LE [BINARY 5] (LINEAR INTERPOLATION END)**  
**LF (FORWARD LIMIT)**  
**LI [BINARY 1] (LINEAR INTERPOLATION DISTANCE)**  
**LL (LIST LABELS)**  
**LM (LINEAR INTERPOLATION MODE)**

**LO (LOCKOUT)**  
**LR (REVERSE LIMIT)**  
**LS (LIST PROGRAM)**  
**LV (LIST VARIABLES)**  
**LZ (LEADING ZERO)**  
**MB (MODBUS)**  
**MC (MOTION COMPLETE - "IN POSITION")**  
**MF (FORWARD MOTION TO POSITION)**  
**MG (MESSAGE)**  
**MM (MASTER MODULUS)**  
**MO (MOTOR OFF)**  
**MR (REVERSE MOTION TO POSITION)**  
**MT (MOTOR TYPE)**  
**NA (NUMBER OF AXES)**  
**NB (NOTCH BANDWIDTH)**  
**NF (NOTCH FREQUENCY)**  
**NO (NO OPERATION)**  
**NZ (NOTCH ZERO)**  
**OB (OUTPUT BIT)**  
**OC (OUTPUT COMPARE)**  
**OE (OFF ON ERROR)**  
**OF (OFFSET)**  
**OP (OUTPUT PORT)**  
**@OUT (STATUS OF DIGITAL OUTPUT)**  
**PA (POSITION ABSOLUTE)**  
**PF (POSITION FORMAT)**  
**PR (POSITION RELATIVE)**  
**QD (DOWNLOAD ARRAY)**  
**QR (DATA RECORD)**  
**QU (UPLOAD ARRAY)**  
**QW (SLAVE RECORD UPDATE RATE)**  
**QZ (RETURN DATA RECORD INFORMATION)**  
**RA (RECORD ARRAY)**  
**RC (RECORD)**  
**RD (RECORD DATA)**  
**RE (RETURN FROM ERROR ROUTINE)**  
**RI (RETURN FROM INTERRUPT ROUTINE)**  
**RL (REPORT LATCHED POSITION)**  
**@RND (ROUND)**  
**RS (RESET)**  
**<CONTROL>R <CONTROL>S (MASTER RESET)**  
**<CONTROL>R <CONTROL>V (REVISION INFORMATION)**  
**SA (SEND COMMAND)**  
**SB (SET BIT)**  
**SC (STOP CODE)**  
**SH (SERVO HERE)**  
**@SIN (SIN)**

**SP (SPEED)**  
**@SQR (SQUARE ROOT)**  
**ST (STOP)**  
**TB (TELL STATUS BYTE)**  
**TC (TELL ERROR CODE)**  
**TD (TELL DUAL ENCODER)**  
**TE (TELL ERROR)**  
**TI (TELL INPUTS)**  
**TIME (TIME OPERAND KEYWORD))**  
**TL (TORQUE LIMIT)**  
**TM (TIME COMMAND)**  
**TP (TELL POSITION)**  
**TR (TRACE)**  
**TS (TELL SWITCHES)**  
**TT (TELL TORQUE)**  
**TV (TELL VELOCITY)**  
**TW (TIMEOUT FOR IN POSITION (MC))**  
**UL (UPLOAD)**  
**VA (VECTOR ACCELERATION)**  
**VD (VECTOR DECELERATION)**  
**VE (VECTOR SEQUENCE END)**  
**VF (VARIABLE FORMAT)**  
**VR (VECTOR SPEED RATIO)**  
**VS (VECTOR SPEED)**  
**VT (VECTOR TIME CONSTANT)**  
**WC (WAIT FOR CONTOUR DATA)**  
**WT (WAIT)**  
**XQ (EXECUTE PROGRAM)**  
**ZA (USER VARIABLES)**  
**ZB (USER VARIABLES)**  
**ZS (ZERO SUBROUTINE STACK)**

## 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-3010, the axis designators a,b,c,d,e,f,g,h are used where x,y,z,w can be used interchangeably with a,b,c,d.

Valid x,y,z,w syntax	
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,,f	Specify a,d and f 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-3010 uses ABCDEFGH axis designators where XYZW can be used interchangeably with ABCD.

Valid XYZW syntax	
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 ABFG	Servo Here, A,B,F,G axes

Where X,Y,Z and W specify axes.

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

FUNCTION: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 command "OE"). AB aborts motion on all axes in motion and cannot stop individual axes. If a multi-axis system is configured (distributed control) the AB command will abort all axes if issued to the master.

**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"	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

## FUNCTION: 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

**FUNCTION:** Acceleration

[Motion]

**DESCRIPTION:**

The Acceleration (AC) command sets the linear acceleration rate for independent moves, such as PR, PA and JG 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 command is used to specify the deceleration rate.

**ARGUMENTS:** AC x, y, z, w or ACX=x or AC a, b, c, d, e, f, g, h where

x, y z, w, or a, b, c, d, e, f, g, h are unsigned integers

“?” returns the acceleration value

**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		

**OPERAND USAGE:**

\_ACn contains the value of acceleration in counts/sec<sup>2</sup> where n is an axis letter

**RELATED COMMANDS:**

"DC"	Specifies deceleration rate.
"FA"	Feedforward Acceleration.
"IT"	Smoothing constant - S-curve

**EXAMPLES:**

AC 150000	Set acceleration to 150000 counts/sec <sup>2</sup>
AC ?	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

### FUNCTION: 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

**FUNCTION:** 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.

**Note:** AD will be affected when motion smoothing time constant, IT, is not 1. See IT command for more information.

**ARGUMENTS:** ADx, y, z, w or ADX=x or AD a, b, c, d, e, f, g, h where

x, y, z, w, or a, b, c, d, e, f, g, h 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		

**RELATED COMMANDS:**

"AR"	After relative distance for sequential triggering
------	---

**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.

# AF

**FUNCTION:** Analog Feedback

[Configuration]

**DESCRIPTION:**

The Analog Feedback (AF) command is used to set an axis with analog feedback instead of digital feedback (quadrature/pulse dir). As the analog feedback is decoded by a 12-bit A/D converter, an input voltage of 10 volts is decoded as a position of 2047 counts and a voltage of -10 volts corresponds to a position of -2048 counts.

**ARGUMENTS:** *AF x,y,z,w or AFX=x or AF a,b,c,d,e,f,g,h where*

x,y,z,w or a, b, c, d, e, f, g, h

1 = Enables analog feedback

0 = Disables analog feedback and switches to digital feedback

"?" returns a 0 or 1 which states whether analog feedback is enabled for the specified axes.

**USAGE:**

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

**OPERAND USAGE:**

\_AFn returns the current feedback setting where n is an axis letter

**RELATED COMMANDS:**

"CE"	Configure Encoder
------	-------------------

**EXAMPLES:**

AF 1,0,0,1	Analog feedback on X and W axis
V1 = _AFX	Assign feedback type to variable
AF ?,?,?	Interrogate feedback type of X, Y, Z

# AI

**FUNCTION:** 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 n 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.

**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[n]	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/sec <sup>2</sup>
PR 400	Specify position
BG	Begin motion
AI+ 7; AI- 7	Wait for falling edge on input 7
EN	End Program

**HINT:** The AI command actually halts execution until specified input is at desired logic level. Use the conditional Jump command (JP) or input interrupt (II) if you do not want the program sequence to halt.

# AL

FUNCTION: 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 upon a low going signal on Input 1. 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 returns the captured position value. The CN command will change the polarity of the latch.

**ARGUMENTS:** *ALn*      *where*

n = XYZW or ABCDEFGH for the main encoder latch and

n = SX, SY, SZ, SW or SA, SB, SC, SD, SE, SF, SG, SH for the auxiliary 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		

**OPERAND USAGE:**

*\_ALn* or *\_ALS<sub>n</sub>* contains the state of the latch where n is an axis letter. 0 = not armed, 1 = armed.

**RELATED COMMANDS:**

"RL"	Report Latch
"CN"	Configure

**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

**FUNCTION:** After Move

[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 to verify position error for servos or use the MC trippoint to wait until final position is reached by the servo.

**ARGUMENTS:** *AM XYZWS or ABCDEFGH*                      *where*

X, Y, Z, W or A, B, C, D, E, F, G, H 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		

**RELATED COMMANDS:**

"BG"	_BG returns a 0 if motion complete
"MC"	Actual Motion Complete

**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.

# @AN

## FUNCTION: Read Analog Input

### DESCRIPTION:

@AN returns the value of an analog input as a voltage (+/-10V). Note that the @AN 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. The resolution is 14 bit, or 1.2mV per bit. To read analog inputs from a slave controller, use the SA command.

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

n is an unsigned integer

### USAGE:

While Moving	Yes	Minimum n Value	1
In a Program	Yes	Maximum n Value	2
Not in a Program	Yes	Default n Value	n/a
Can be Interrogated	No	Default Format	10.4
Used as an Operand	Yes		

### EXAMPLES:

#TEST	Program TEST
MG @AN[2]	Display the value of analog input #2 as a voltage
JGX=@AN[2]*10000	Set jog speed according to analog input
SAA="MG", "@AN", "[2]"	Sends command MG @AN[2] to slave on handle A
Analog3=_SAA	Returns slave response to SA command
BGX	Begin Move
EN	End of program

# AO

**FUNCTION:** Analog Out

[I/O]

**DESCRIPTION:**

The AO command sets the analog output voltage of the local analog output or ModBus devices connected via Ethernet.

**ARGUMENTS:**AO *m*, *n* *where*

*m* is either the local analog output voltage ranging from 9.9982 to -9.9982 or the I/O number calculated using the following equations:

$$m = (\text{SlaveAddress} * 1000) + (\text{HandleNum} * 1000) + (\text{Module} - 1) * 4 + (\text{BitNum} - 1)$$

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

*n* = the voltage which ranges from 9.9982 to -9.9982. If *m* is < 1000, *n* is omitted.

**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	Yes, local only		

**OPERAND USAGE:** \_AO returns the value of the local analog output in volts.

**RELATED COMMANDS:**

"SB"	Set Bit
"CB"	Clear Bit
"MB"	Modbus

**EXAMPLES:**

AO -3.4	Sets local analog output to -3.4V
AO 6016, 8.2	Sets analog output on modbus device on handle F to 8.2V
SAA="AO",2.7	Set analog output of slave on handle A to 2.7V
SAA="MG", "_AO"	Send command MG_AO to slave controller on handle A
VAR1=_SAA	Store the returned value to VAR1

# AP

**FUNCTION:** 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, e, f, g, h where

x, y, z, w, or a, b, c, d, e, f, g, h 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		

**RELATED COMMANDS:**

"AR"	Trippoint for relative distances
"MF"	Trippoint for forward motion

**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 command to measure incremental distances.

# AR

**FUNCTION:** After Relative Distance

[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, e, f, g, h where

x, y z, w, or a, b, c, d, e, f, g, h 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		

**RELATED COMMANDS:**

"AP"	Trippoint for 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

FUNCTION: 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 ABCDEFGH*                      *where*

X, Y, Z, W or A, B, C, D, E, F, G, H 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		

**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

### FUNCTION: 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

FUNCTION: At 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	-2147483647
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

### FUNCTION: Arc Tangent Function

**DESCRIPTION:**

@TAN returns the arc tangent, in degrees, 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	-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 @ATAN[VAR1]	Set variable
VAR1=.707	Display the arc sine of .707
VAR2=@ATAN[VAR1]+5	Perform calculation
EN	End of program

# BG

**FUNCTION:** 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 XYZWS or ABCDEFGH*                      *where*

X, Y, Z, W, S or A, B, C, D, E, F, G, H specify the axis or sequence. 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		
Can be Interrogated	No		
Used as an Operand	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 complete
"ST"	Stop motion

**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.*

# BL

**FUNCTION:** Reverse Software 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, e, f, g, h where*

x, y z, w, or a, b, c, d, e, f, g, h are signed integers

“?” returns the reverse software limit value

-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		

**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

FUNCTION: Burn

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

**ARGUMENTS:** None

**USAGE:**

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

# BP

**FUNCTION:** Burn Program

[General]

**DESCRIPTION:**

The BP command saves the application program 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

FUNCTION: Burn Variables and Array

[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		

# CB

**FUNCTION:** Clear Bit

[I/O]

**DESCRIPTION:**

The CB command clears a bit on the output port by setting it to logic zero. Slave controller or 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. A LEGEND-MC controller has 4 digital outputs plus applicable I/O connected by Modbus.

**DISTRIBUTED CONTROL:**

Handle	Command	Handle	Command
A	CB101 ~ CB104	E	CB501 ~ CB504
B	CB201 ~ CB204	F	CB601 ~ CB604
C	CB301 ~ CB304	G	CB701 ~ CB704
D	CB401 ~ CB404	H	CB801 ~ CB804

**MODBUS:**

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

$$n = (\text{SlaveAddress} * 1000) + (\text{HandleNum} * 1000) + ((\text{Module}-1) * 4) + (\text{Bitnum}-1)$$

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"	Define all outputs

**EXAMPLES:**

CB 1	Clear output bit 1
CB 2	Clear output bit 2
CB 3	Clear output bit 3
CB 602	Clear output 2 on slave controller on handle F

# CD

**FUNCTION:** 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 the Contour Mode (CM).

**ARGUMENTS:** *CD x, y, z, w or CDX=x or CD a, b, c, d, e, f, g, h where*

x, y z, w, or a, b, c, d, e, f, g, h 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		

**RELATED COMMANDS:**

"CM"	Contour Mode
"WC"	Wait for Contour
"DT"	Time Increment

**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

**FUNCTION:** 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.

**ARGUMENTS:** *CE x, y, z, w or CEX=x or CE a, b, c, d, e, f, g, h where*

x, y, z, w, or a, b, c, d, e, f, g, h 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.

“?” returns the encoder inputs

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

**OPERAND USAGE:**

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

**RELATED COMMMANDS:**

"MT"	Specify motor type
------	--------------------

**EXAMPLES:**

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

# CF

## FUNCTION: Configure Messages

[Configuration]

**DESCRIPTION:**

Sets the controller's default port for unsolicited messages. By default, the LEGEND-MC 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 command with no port designation specified.

**ARGUMENTS:** *CF n where*

n is A thru 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".

# CH

FUNCTION: Connect Handle

[General]

**DESCRIPTION:**

The CH command is used to associate master and slave controllers in a distributed control system. The master controller must associate one Ethernet handle for sending commands to each slave, and one Ethernet handle for receiving status information from each slave. Note that these handles must first be opened before assigning them with this command, see the command IH.

**ARGUMENTS:** CHx=h1,h2 where

x is X, Y, Z, W or A, B, C, D, E, F, G, H.

h1 is the handle (character) to be used to send commands to the slave controller.

h2 is the handle (character) to be used for receiving status from the slave controller.

**USAGE:**

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

**RELATED COMMANDS:**

"IH"	Set Internet Handles
"NA"	Set Number of Axes for Distributed Control System
"QW"	Set Slave Data record Update Rate

**EXAMPLE:**

CHY=A,B	Using one LEGEND-MC as a master and one LEGEND-MC as a slave under TCP/IP. This command assigns a slave, identified by the Y axis designator, with Handle A for commands and Handle B for status returned from the slave.
---------	---

An example subroutine demonstrating how to release handles:

#RELEASE	Release handles
MG "Releasing..."	Diagnostic message
IHE=>-2	Release handle E
#WAIT; JP #WAIT,_IHE2<>0	Wait until handle is released
IHF=>-2	Release handle
#WAIT1; JP #WAIT1,_IHF2<>0	Wait until handle is released
EN	End subroutine

An example subroutine demonstrating how to assign handles:

#ASSIGN	Assign handles
MG "Assigning..."	Diagnostic message
IHE=192,168,3,104>2	Must let first IH command succeed before setting another IH
#WAITA; JP #WAITA,_IHE2<>-2	Wait until handle is connected
IHF=192,168,3,104<504>2	Assign response handle to slave
#WAITB; JP #WAITB,_IHF2<>-2	Wait until handle is connected
EN	End subroutine

An example subroutine demonstrating how to connect handles:

#CONNECT	Connect to slave
MG "Connecting..."	Diagnostic message
NA2	Set two axis configuration
CHY=E,F	Connect the handles
QWF=4	Set slave response update (don't need to set QW for commands on handle A)
EN	End subroutine

# CM

**FUNCTION:** Contouring 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 ABCDEFGH*

CM? Returns a 1 if the contour buffer is full, and 0 if the contour buffer is empty.

**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		

**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"	Time Increment

**EXAMPLES:**

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

# CN

**FUNCTION:** 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* where

m, n, o 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

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

**USAGE:**

While Moving	Yes	Default Value	-1.-1.-1
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.

**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

## FUNCTION: 2's Complement

### 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

## FUNCTION: Cosine

### 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

# CS

FUNCTION: Clear Sequence

[General]

**DESCRIPTION:**

The CS command will remove VP or LI commands stored in a motion sequence. 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 correctly specified VP or LI 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 the Linear Mode, LM, Vector Mode, VM and Contour Mode, CM .

**ARGUMENTS:** none

**USAGE:**

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

**EXAMPLES:**

#CLEAR	Label
VP 1000,2000	Vector position
VP 4000,8000	Vector position
CS	Clear vectors
VP 1000,5000	New vector
VP 8000,9000	New vector
VE	End sequence
BGS	Begin motion sequence
EN	End of program

# CW

**FUNCTION:** Copyright information / Data Adjustment Bit On/Off [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.

**ARGUMENTS:** CW n,m 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.
- “?” returns the copyright information for the controller

m is 0 or 1 (optional)

- 0 Causes the controller to pause program execution when output FIFO is full until FIFO no longer full.
- 1 Causes the controller to continue program execution when output FIFO is full - output characters after FIFO is full will be lost.

**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:**

\_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 may sometimes 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 status can be stored in EEPROM.**

# DA

**FUNCTION:** Deallocate Variables & Arrays

[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 14. If one array is defined, the operand \_DA will return 13.

**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

**FUNCTION:** Deceleration

[Motion]

**DESCRIPTION:**

The Deceleration command (DC) sets the linear deceleration rate for independent moves such as PR, PA and JG 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, e, f, g, h where*

x, y z, w, or a, b, c, d, e, f, g, h are unsigned integers

“?” returns the deceleration value

**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		

**OPERAND USAGE:**

DCn contains the deceleration rate in counts/sec<sup>2</sup> where n is an axis letter.

**RELATED COMMANDS:**

"AC"	Acceleration
"PR"	Position Relative
"SP"	Speed
"JG"	Jog
"BG"	Begin
"IT"	Smoothing constant - S-curve

**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

**FUNCTION:** Dual (Auxiliary) Encoder Position

[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, e, f, g, h where*

*x, y, z, w, or a, b, c, d, e, f, g, h are signed integers*

*“?” returns the position of the auxiliary encoder*

**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		

**OPERAND USAGE:**

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

**EXAMPLES:**

<i>:DE 0</i>	<i>Set the auxiliary encoder position to 0</i>
<i>:DE?</i>	<i>Return auxiliary encoder positions</i>
<i>:DUALX=_DE</i>	<i>Assign auxiliary encoder position of X-axis to the variable DUALX</i>

**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

**FUNCTION:** 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 LEGEND-MC RAM. The data is entered beginning at line 0. If there are too many lines or too many characters per line, the LEGEND-MC 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 LEGEND-MC program in RAM.

**ARGUMENTS:** *DL n*

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 126.

**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

**FUNCTION:** Dimension

[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.

**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 8000.

DM? Returns the number of array elements available.

**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 8000. If an array of 100 elements is defined, the operand \_DM will return 7900.

**RELATED COMMANDS:**

"DA"	Deallocate Array
------	------------------

**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

**FUNCTION:** 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 and RP values.

**ARGUMENTS:** *DP x, y, z, w or DPX=x or DP a, b, c, d, e, f, g, h where*

x, y z, w, or a, b, c, d, e, f, g, h are signed integers

“?” returns the current position of the motor

**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		

**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.
:DP ?	
-0050000	Returns the motor position

# DT

**FUNCTION:** 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.

DT? Returns the value for the time interval for contour mode.

**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 next data

**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

**FUNCTION:** Dual Velocity (Dual Loop)

[Setting]

**DESCRIPTION:**

The DV function changes the operation of the PID servo loop. It causes the KD (derivative) 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, e, f, g, h where*

x, y, z, w, or a, b, c, d, e, f, g, h are unsigned integers

n may be 0 or 1. 0 disables the function. 1 enables the dual loop.

“?” returns a 0 if dual velocity mode is disabled and 1 if enabled for the specified axis.

**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		

**OPERAND USAGE:**

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

**RELATED COMMANDS:**

"KD"	Damping 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

FUNCTION: ECAM master

[Setting]

**DESCRIPTION:**

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

**ARGUMENTS:** EA x where

X or SX

**USAGE:**

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

**RELATED COMMANDS:**

"EB"	Enable ECAM
"EC"	Set ECAM table index
"EG"	Engage ECAM
"EM"	Specify ECAM cycle
"EP"	Specify ECAM table intervals & starting point
"EQ"	Disengage ECAM
"ET"	ECAM table

**EXAMPLES:**

EASX	Select auxiliary encoder as the master for ECAM
------	---

# EB

## FUNCTION: 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.

EB? Returns a 0 if ECAM is disabled and 1 if enabled.

**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"	Specify Ecam Cycle
"EP"	CAM table intervals & starting point
"MM"	Master's Modulus

**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

FUNCTION: 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 and is most useful when sending commands in binary. See the command, ET.

**ARGUMENTS:** *EC n where*

n is an integer between 0 and 256.

n = ? Returns the current value of the index into the ECAM table.

**USAGE:**

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

**OPERAND USAGE:**

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

**RELATED COMMANDS:**

"EA"	Choose ECAM master
"EB"	Enable ECAM
"EG"	Engage ECAM
"EM"	Specify ECAM cycle
"EP"	Specify ECAM table intervals & starting point
"EQ"	Disengage ECAM
"ET"	ECAM table

**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

FUNCTION: Edit

[General]

**DESCRIPTION:**

Using Yaskawa YTerm Software or any other terminal emulator: 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:

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

**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.

**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

## FUNCTION: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 n where*

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

“?” returns 1 if specified axis is engaged and 0 if disengaged

**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		

**OPERAND USAGE:**

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

**RELATED COMMANDS:**

“EB”	Enable Ecam
”EQ”	Ecam quit

**EXAMPLES:**

EG 700	Engages slave at master position 700.
B = _EP	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

**FUNCTION:** Else Function for use with IF Conditional Statement [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)	2 <sup>nd</sup> IF conditional statement executed if 1 <sup>st</sup> IF conditional true
MG "INPUT 1 AND INPUT 2 ARE ACTIVE"	Message to be executed if 2 <sup>nd</sup> IF conditional is true
ELSE	ELSE command for 2 <sup>nd</sup> IF conditional statement
MG "ONLY INPUT 1 IS ACTIVE"	Message to be executed if 2 <sup>nd</sup> IF conditional is false
ENDIF	End of 2 <sup>nd</sup> conditional statement
ELSE	ELSE command for 1 <sup>st</sup> IF conditional statement
MG"ONLY INPUT 2 IS ACTIVE"	Message to be executed if 1 <sup>st</sup> IF conditional statement
ENDIF	End of 1 <sup>st</sup> conditional statement

# EM

FUNCTION: 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 n where*

n is the net change in the slave axis.

**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		

**OPERAND USAGE:**

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

**RELATED COMMANDS:**

"EP"	CAM table intervals & starting point
"ET"	Electronic CAM table
"EB"	Enable Ecam

**EXAMPLES:**

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

# EN

FUNCTION: 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 command, the EN command ends the subroutine and returns program flow to the point just after the JS command.

The EN command is also used to end the automatic subroutines #MCTIME and #CMDERR.

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 subroutine
"RI"	Return from interrupt subroutine

**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

**FUNCTION:** End of IF Conditional Statement

[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 command
"JS"	Jump to subroutine command

**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

**FUNCTION:** 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.

**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

**FUNCTION:** Cam Table Intervals and Starting 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 257 points may be specified using the ET 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"	Enable Ecam
"EG"	Engage Ecam
"EM"	Specify Ecam Cycle
"EQ"	Ecam quit
"ET"	Electronic CAM table

**EXAMPLES:**

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

# EQ

FUNCTION: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.

“?” contains a 1 if engage command issued and slave is waiting to engage, 2 if disengage command issued and slave is waiting to disengage, and 0 if ECAM engaged or 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		

**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"	Enable Ecam
"EG"	Engage Ecam
"EM"	Specify Ecam Cycle
"EP"	CAM table intervals & starting point
"ET"	Electronic CAM 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

**FUNCTION:** 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.

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

x, y z, w, or a, b, c, d, e, f, g, h are unsigned integers

“?” returns the value of the Error limit.

**USAGE:**

While Moving	Yes	Minimum Value	1
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		

**OPERAND USAGE:**

ERn contains the value of the Error limit where n is an axis letter.

**RELATED COMMANDS:**

#POSERR	Automatic Error Subroutine
---------	----------------------------

**EXAMPLES:**

ER 200	Set the error limit to 200
ER ?	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.*

# ET

## FUNCTION:Electronic Cam 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 command.

**ARGUMENTS:** *ET [n] = m where*

n is an integer.

m is an integer.

**USAGE:**

While Moving	Yes	Minimum n Value	0
In a Program	Yes	Maximum n Value	256
Command Line	Yes	Default n Value	n/a
Can be Interrogated	No	Minimum m Value	-2147438648
Used as an Operand	No	Maximum m Value	2147438647
		Default m Value	n/a

**RELATED COMMANDS:**

"EB"	Enable Ecam
"EG"	Engage Ecam
"EM"	Specify Ecam Cycle
"EP"	Specify Ecam intervals and starting point
"EQ"	Ecam quit

**EXAMPLES:**

ET [7] = 1000	Specifies the position of the slave that must be synchronized with the eighth increment of the master.
---------------	--

# FA

**FUNCTION:** 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.

$$\text{Acceleration Feedforward Bias} = \text{FA} \cdot \text{AC} \cdot 1.5 \cdot 10^{-7}$$

$$\text{Deceleration Feedforward Bias} = \text{FA} \cdot \text{DC} \cdot 1.5 \cdot 10^{-7}$$

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, e, f, g, h where

x, y z, w, or a, b, c, d, e, f, g, h are unsigned numbers

“?” returns the value of the feedforward acceleration coefficient.

FA has a resolution of .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		

**OPERAND USAGE:**

\_FA<sub>n</sub> 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
FA ?	The effective bias will be 0.75V (10 * 500000 * 1.5 * 10 <sup>-7</sup> )
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

**FUNCTION:** 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 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 ABCDEFGH*

**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:**

"FI"	Find Index
"HM"	Home
"BG"	Begin
"AC"	Acceleration Rate
"DC"	Deceleration Rate
"SP"	Speed for search

**EXAMPLES:**

FE	Set find edge mode
BG	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 after each of these commands. Use HM (Home) to search for both the Home input and the Index.

# FI

**FUNCTION:** Find Index

[Motion]

**DESCRIPTION:**

The FI and BG commands move 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 command.

**ARGUMENTS:** *FI XYZW or ABCDEFGH*

**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:**

"FE"	Find Edge
"HM"	Home
"BG"	Begin
"AC"	Acceleration Rate
"DC"	Deceleration Rate
"JG"	Speed for search

**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 to search for the Home input. Use HM (Home) to search for both the Home input and the Index. Remember to specify BG after each of these commands.

# FL

**FUNCTION:** Forward Software 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, e, f, g, h where*

x, y, z, w, or a, b, c, d, e, f, g, h are unsigned numbers

“?” returns the value of the forward limit switch

**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		

**OPERAND USAGE:**

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

**RELATED COMMANDS:**

"BL"	Reverse 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

**FUNCTION:** Fraction

**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

**FUNCTION:** 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.

$$\text{Velocity feedforward bias} = 1.22 \cdot 10^{-6} \cdot \text{FV} \cdot \text{Velocity [in ct/s]}.$$

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, e, f, g, h where*

x, y, z, w, or a, b, c, d, e, f, g, h are unsigned numbers

“?” returns the feedforward velocity for the specified axis.

**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		

**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 ( $1.22 \times 10^{-6} \times 10 \times 30000$ ).
FV ?	Return the value
010	

# GA

**FUNCTION:** Master Axis for Gearing

[Setting]

**DESCRIPTION:**

The GA command specifies the master axis for electronic gearing.

The master axis is the auxiliary encoder on the LEGEND-MC. The slave ratio is specified with the GR command and gearing is turned off by the command GR0.

**ARGUMENTS:** GADX

**USAGE:**

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

**RELATED COMMANDS:**

"GR"	Gear Ratio
------	------------

**EXAMPLES:**

#GEAR	Gear program
GADX	Specify auxiliary encoder axis as master
GR -2.5	Specify X ratio

# GR

FUNCTION: Gear Ratio

[Motion]

**DESCRIPTION:**

GR specifies the Gear Ratio for the slave axis in electronic gearing mode. The master axis for the LEG-  
END-MC is specified with the GA 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

“?” returns the value of the gear ratio

**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		

**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

# HM

FUNCTION: 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 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 ABCDEFGH*

**USAGE:**

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

**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 Home
"FI"	Find Index Only
"FE"	Find Home Only

**EXAMPLES:**

HM	Set Homing Mode
BG	Begin Homing

**HINT:** You can customize homing sequence by using the FE (Find Home Sensor only) and FI (Find Index only) commands.

# HX

**FUNCTION:** Halt Execution

[Program Flow]

**DESCRIPTION:**

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

**ARGUMENTS:** *HX n where*

n is 0 or 1 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,1	Execute program #B, thread one
HX0	Halt thread zero
HX1	Halt thread one

# IA

**FUNCTION:** IP 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.

>u specifies the multicast IP address where u is an integer between 0 and 63.

IA? will return the IP address of the controller

**USAGE:**

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

**OPERAND USAGE:**

\_IA0contains the IP address representing a 32 bit signed number (Two's complement)

\_IA1contains the value for t (retry time)

\_IA2contains the number of available handles

\_IA3contains the number of the handle using this operand where the number is 0 to 7. 0 represents handle A, 1 handle B, etc.

**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

**FUNCTION:** IF conditional statement

[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

## FUNCTION: Open Internet Handle

[Setting]

### DESCRIPTION:

The IH command is used when the LEGEND-MC is operated as a network master. This command opens a handle and connects to a slave.

Each 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 isn't 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 or IHh= >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
?			Returns the IP address as four 1 byte numbers

### 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

### FUNCTION: 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 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	7	Zero disables the interrupts. 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	7	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	127	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,, 15	Equivalent to binary 00001111, inputs #1 through #4 will be enabled for interrupt.
p	1	127	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.

**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"	Trippoint for input

**EXAMPLES:**

#A	Program A
II 1	Specify interrupt on input 1

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

## FUNCTION: 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 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 and KP terms remain active in any case.

### ARGUMENTS: IL x, y, z, w or ILX=x or IL a, b, c, d, e, f, g, h where

x, y z, w, or a, b, c, d, e, f, g, h are unsigned integers

“?” returns the value of the integrator limit

### 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		

### 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

## FUNCTION: Input Variable

[General]

### DESCRIPTION:

The IN command allows a variable to be input from the serial port or Ethernet. 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}.**

**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

**FUNCTION:** Status of Digital 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 output on the controller to be cleared (set to 0). The first output on the controller is denoted as output 1. A LEGEND-MC controller has 4 digital outputs plus applicable I/O connected by Modbus.

**DISTRIBUTED CONTROL:**

Handle	Command	Handle	Command
A	@IN[101] ~ @IN[104]	E	@IN[501] ~ @IN[504]
B	@IN[201] ~ @IN[204]	F	@IN[601] ~ @IN[604]
C	@IN[301] ~ @IN[304]	G	@IN[701] ~ @IN[704]
D	@IN[401] ~ @IN[404]	H	@IN[801] ~ @IN[804]

**MODBUS:**

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

$$n = (\text{SlaveAddress} * 1000) + (\text{HandleNum} * 1000) + ((\text{Module} - 1) * 4) + (\text{BitNum} - 1)$$

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

### FUNCTION: Integer

#### 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

**FUNCTION:** 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. 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 n and BG 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 n command will cause the motor to instantly try to servo to a position n from the present instantaneous position. The SP and AC 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 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, e, f, g, h where

x, y, z, w, or a, b, c, d, e, f, g, h are unsigned integers

“?” returns the current position

**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		

**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

**FUNCTION:** Independent Time Constant - Smoothing Function [Motion]

**DESCRIPTION:**

The IT command filters the acceleration and deceleration functions in independent moves of JG, PR, PA 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 and AD. 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.

**ARGUMENTS:** *IT x, y, z, w or ITX=x or IT a, b, c, d, e, f, g, h where*

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

“?” returns the value of the independent time constant.

**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		

**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
IT ?	Return independent time constant
0.8	

# JG

**FUNCTION:** 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, e, f, g, h where*

x, y z, w, or a, b, c, d, e, f, g, h are signed integers

“?” returns the absolute value of the jog speed.

**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		

**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

**FUNCTION:** 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

**FUNCTION:** 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 command is encountered (End of subroutine), 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

**FUNCTION:** Derivative Constant

[Tuning]

**DESCRIPTION:**

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

$$D(z) = 4 \cdot KP + 4 \cdot KD(z-1)/z + KIz/2 (z-1)$$

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, e, f, g, h where*

x, y z, w, or a, b, c, d, e, f, g, h are unsigned integers

“?” returns the value of the derivative constant

**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		

**OPERAND USAGE:**

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

**RELATED COMMANDS:**

"KP"	Proportional Constant
"KI"	Integral

**EXAMPLES**

KD 100	Specify KD
KD ?	Return KD
0100.00	

# KI

**FUNCTION:** Integrator

[Tuning]

**DESCRIPTION:**

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

$$D(z) = 4 \cdot KP + 4 \cdot KD(z-1)/z + KI z/2(z-1)$$

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, e, f, g, h where*

x, y z, w, or a, b, c, d, e, f, g, h are unsigned integers

“?” returns the value of the integrator

**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		

**OPERAND USAGE:**

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

**RELATED COMMANDS:**

"KP"	Proportional Gain
"KD"	Derivative
"IL"	Integrator Limit

**EXAMPLES:**

KI 12	Specify integral
KI ?	Return value
0012	

# KP

**FUNCTION:** Proportional Constant

[Tuning]

**DESCRIPTION:**

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

$$D(z) = 4 \cdot KP + 4 \cdot KD(z-1)/z + KI z/2(z-1)$$

For further details see the section Theory of Operation.

**ARGUMENTS:** *KP x, y, z, w or KPX=x or KP a, b, c, d, e, f, g, h where*

x, y z, w, or a, b, c, d, e, f, g, h are unsigned integers

“?” returns the value of the proportional constant

**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		

**OPERAND USAGE:**

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

**RELATED COMMANDS:**

"KP"	Proportional Gain
"KI"	Integral constant

# LA

**FUNCTION:** 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 Variable

**EXAMPLES:**

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

# LE

**FUNCTION:** Linear Interpolation End

[Motion]

**DESCRIPTION:** LE

Signifies the end of a linear interpolation sequence. It follow the last LI specification in a linear sequence. The LE command signifies the controller issues commands to decelerate the motor to a stop.

**ARGUMENTS:**

n=? Returns the total vector move length in encoder counts for the coordinate system.

**USAGE:**

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

**OPERAND USAGE:**

\_LE contains the total vector move length in encoder counts.

**RELATED COMMANDS:**

"LI"	Linear Distance
"BGS"	BGS - Begin Sequence
"LM"	Linear Interpolation Mode
"VS"	Vector Speed
"VA"	Vector Acceleration
"VD"	Vector Deceleration
"PF"	Position Formatting

**EXAMPLES:**

LM XY	Specify linear interpolation mode for X and Y axes
LI 100, 200	Specify linear distance
LE	End linear move
BGS	Begin sequence

## **\_LF\***

**FUNCTION:** Forward Limit Switch Operand (Keyword)

[Status]

**DESCRIPTION:** *\_LF XYZW or LF ABCDEFGH*

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: 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

**\*This is an Operand - Not a command.**

# LI

## FUNCTION: Linear Interpolation Distance

[Motion]

### DESCRIPTION:

The LI x, y 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 Linear End (LE) command must be given after the last LI specification in a sequence. This command tells the controller to decelerate to a stop at the last LI command. It is the responsibility of the user to keep enough LI segments in the controller's sequence buffer to ensure continuous motion.

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. It should be noted that the controller computes the vector speed based on the axes specified in the LM mode. For example, LM XY designates linear interpolation for the X and Y axes. The speed of these axes will be computed from  $VS^2 = XS^2 + YS^2$  where XS and YS are the speed of the X and Y axes. The controller always uses the axis specifications from LM, not LI, to compute the speed. The parameters o and p are optional and can be used to define the vector speed that is attached to the motion segment.

Linear Interpolation is useful for making blended move profiles.

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

Argument	Min	Max	Note	Example	Meaning
n	-8388607	8388607	The incremental move distance.	LI 500	500 encoder count move on the X 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	12000000	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		

### RELATED COMMANDS:

"LE"	Linear End
"BGS"	BGS - Begin Sequence
"LM"	Linear Interpolation Mode
"CS"	Clear Sequence
"VS"	Vector Speed
"VA"	Vector Acceleration

"VD"	Vector Deceleration
------	---------------------

**EXAMPLES:**

LM XY	Specify linear interpolation mode for X and Y axes
LI 1000, 2000	Specify linear distance
LE	End linear move
BGS	Begin sequence

**LL****FUNCTION:** 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.

**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
------	----------------

**EXAMPLES:**

: LL	
# FIVE	
# FOUR	
# ONE	
# THREE	
# TWO	

# LM

**FUNCTION:** Linear Interpolation Mode

[Setting]

**DESCRIPTION:**

The LM command specifies the linear interpolation mode and specifies the axes for linear interpolation. LI commands are used to specify the travel distances for linear interpolation. The LE command specifies the end of the linear interpolation sequence. Several LI 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 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 XY designates linear interpolation for the X and Y axes. The speed of these axes will be computed from  $VS^2=XS^2+YS^2$ , where XS and YS are the speed of the X and Y axes. The controller always uses the axis specifications from LM, not LI, to compute the speed.

**ARGUMENTS:** LM XYZW or ABCDEFGH

**USAGE:**

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

**OPERAND USAGE:**

\_LMx contains the number of spaces available in the sequence buffer for the coordinate system.

**RELATED COMMANDS:**

"LE"	Linear end
"LI"	Linear Distance
"VA"	Vector acceleration
"VS"	Vector Speed
"VD"	Vector deceleration
"CS"	_CS - Sequence counter

**EXAMPLES:**

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

# LO

**FUNCTION:** Lockout

[Configuration]

**DESCRIPTION:**

The LO command is used to lock-out a particular handle or serial port with the master controller on a distributed control system. This function ignores all data received to the master on the specified communication channel.

**ARGUMENTS:** *LO h,n*                      *where*

h is the handle, A thru F, or the letter S for the serial port. This identifies the communication channel to be locked out.

n = 1 or no argument to enable the lockout

n = -1 to remove the lockout

**USAGE:**

While Moving	Yes	Default Value	n/a
In a Program	Yes	Default Format	n/a
Command Line	Yes	Default n value	1

**OPERAND USAGE:**

\_LOh contains the state of the lockout for handle A - F or S.

**RELATED COMMANDS:**

"CH"	Connect to Internet Handles for slaves
"IH"	Set Internet Handles
"NA"	Set number of axes for distributed control system
"QW"	Set slave data record update rate
"SA"	Send command to slave

**EXAMPLES:**

LOS	Lockout information received from the serial port
WT10000	Wait 10 seconds
LOS,-1	Re-enable the serial port

## **\_LR\***

**FUNCTION:** Reverse Limit Switch Operand (Keyword)

[Status]

**DESCRIPTION:** *\_LR XYZW or ABCDEFGH*

\*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: 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

**\*This is an Operand - Not a command.**

# LS

**FUNCTION:** 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 500, or labels. n is the first line to be listed, m is the last.

**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 <ctrl> Q prior to giving the LS command.

# LV

**FUNCTION:** List Variables

[General]

**DESCRIPTION:**

The LV command returns a listing of all of the program labels 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

**FUNCTION:** Inhibit 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

LZ? Returns the state of the LZ function.

**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		

**EXAMPLES:**

TE	Tell error
0004	
LZ 1	Inhibit leading zeros
TE	Tell error
4	

# MB

FUNCTION: 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 LEGEND-MC 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

# MC

**FUNCTION:** Motion Complete - "In Position"

[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 is completed and the encoder reaches or passes the specified target position. TW sets the timeout to declare an error if the encoder is not in position within a specified time. If a timeout occurs, the trippoint will clear and the stopcode (SC command) will be set to 99. The application program will jump to the special label #MCTIME, if included in your program.

**ARGUMENTS:** MC XYZWS or ACDEFGH

where

X, Y, Z, W, S specify the X, Y, Z, or W axis or sequence. No argument specifies that motion on all axes is complete.

**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:**

"BG"	Begin
"AM"	After Move
"TW"	Timeout

**EXAMPLES:**

#MOVE	Program MOVE
PR 5000	Position relative moves
BG	Start the axis
MC	After the move is complete
SB1	Set output 1 to logic 1
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

**FUNCTION:** Forward Motion to Position

[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, e, f, g, h where*

x, y z, w, or a, b, c, d, e, f, g, h 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		

**RELATED COMMANDS:**

"AR"	Trippoint for after Relative Distance
"MR"	Reverse motion to position
"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

**FUNCTION:** 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}*

"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.

{Sn} Display variable as a string of length n where n is 1 thru 6

{N} Suppress carriage return line feed.

{Ex} For Ethernet and 'x' specifies the Ethernet handle (A,B,C,D,E,F or H).

**Note: Multiple text, variables, and ASCII characters may be used, each must be separated by a comma.**

**Note: The order of arguments is not important.**

**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		

**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.

# MM

FUNCTION: Master's Modulus

[Setting]

**DESCRIPTION:**

The MM command is part of the ECAM mode. The MM command replaces the master modulus setting of the EM command. This allows camming with the auxiliary encoder as the master.

**Note: This command does not work if entered as MMX=nnnn**

**ARGUMENTS: MMx where**

where x is the value of the master modulus in encoder counts.

**USAGE:**

While Moving	No	Minimum value	1
In a Program	Yes	Maximum value	2147483647
Not In a Program	Yes	Default Value	n/a
Can be Interrogated	No	Default Format	8.0
Used as an Operand	Yes		

**OPERAND USAGE:**

\_MMx contains the master modulus

**RELATED COMMANDS:**

"EA"	Select master cam axis
"EP"	Define cam table intervals and start point
"ET"	Cam table entries for the slave axis
"EB"	Enable ECAM mode

**EXAMPLES:**

EADX	Select Auxiliary X axis as Ecam master
MM 30500	Set master modulus
EM 20000	Set main X axis slave modulus
MG_MM	Return master modulus

# MO

FUNCTION: 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 Servo Here command (SH). This command is not allowed while the servo is commanded in motion. Use the ST command first in that case.

**ARGUMENTS:** *MO XYZW or ABCDEFGH*

“?” returns the state of the motor for the specified axis.

**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		

**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
------	------------

**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

**HINT:** *The MO command is useful for positioning the motors by hand. Turn them back on with the SH command.*

# MR

**FUNCTION:** Reverse Motion to Position

[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, e, f, g, h where*

*x, y, z, w, or a, b, c, d, e, f, g, h 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		

**RELATED COMMANDS:**

"AR"	Trippoint for after Relative Distance
"MF"	Forward motion to position
"AP"	Trippoint After absolute position

**EXAMPLES:**

#TEST	Program B
DPO	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

**FUNCTION:** 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. The polarity reversal inverts the analog signals.

**ARGUMENTS:** *MT x, y, z, w or MTX=x or MT a, b, c, d, e, f, g, h* where

x, y z, w, or a, b, c, d, e, f, g, h have one of the following values

1 Servo motor (rotary motor moves counterclockwise when viewing shaft end of motor)

-1 Servo motor reversed polarity

“?” returns the value of the motor type

**USAGE:**

While Moving	Yes	Default Value	1
In a Program	Yes	Default Format	1.0
Command Line	Yes		
Can be Interrogated	Yes		
Used as an Operand	Yes		

**OPERAND USAGE:**

\_MTn contains the value of the motor type where n is an axis letter.

**EXAMPLES:**

MT 1	Configure x as servo
MT ?	Interrogate motor type
V=_MT	Assign motor type to variable

# NA

**FUNCTION:** Number of Axes

[Configuration]

**DESCRIPTION:**

NA defines the total number of axes used in a distributed network control system. This command is used on the master controller. For example; using 3 LEGEND-MC controllers. The command NA3 would be given to the master controller.

**ARGUMENTS:** NA *n*      *where*

*n* is an integer. this number represents the number of axes in a distributed control system.

**USAGE:**

While Moving	Yes	Minimum Value	1
In a Program	Yes	Maximum Value	5
Command Line	Yes	Default Format	---
Can be Interrogated	Yes		
Used as an Operand	Yes		

**OPERAND USAGE:**

\_NA contains the contains the number of axes.

**RELATED COMMANDS:**

"CH"	Connect to the internet handles for slave operation
"IH"	Set internet handles
"QW"	Set Slave Data Record Update Rate

**EXAMPLES:**

NA2	Command given to an LEGEND-MC acting as a multi-axis network master with one slave.
-----	---

# NB

**FUNCTION:** 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, e, f, g, h* where

*x, y, z, w, or a, b, c, d, e, f, g, h* are unsigned integers

**USAGE:**

While Moving	Yes	Minimum value	0
In a Program	Yes	Maximum value	_TM/16
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 Zeros

**EXAMPLES:**

NBX = 10	Sets the real part of the notch pole to 10/2 Hz
NOTCH = _NBX	Sets the variable "NOTCH" equal to the notch bandwidth value for the X axis

# NF

**FUNCTION:** Notch Frequency

[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, e, f, g, h where*

x, y, z, w, or a, b, c, d, e, f, g, h are unsigned integers

“?” returns the value of the Notch filter for the specified axis.

**USAGE:**

While Moving	Yes	Minimum value	0
In a Program	Yes	Maximum value	_TM/4
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

**FUNCTION:** 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.

**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		

**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

FUNCTION: 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, e, f, g, h where*

x, y, z, w, or a, b, c, d, e, f, g, h are unsigned integers

“?” returns the value of the Notch filter zero for the specified axis.

**USAGE:**

While Moving	Yes	Minimum value	0
In a Program	Yes	Maximum value	_TM/16
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

**FUNCTION:** Output Bit

[I/O]

**DESCRIPTION:**

The OB n, logical expression command defines output bit n = 1 through 4 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 4 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 POS 1 is non-zero, Bit 1 is high.
	If POS 1 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

# OC

**FUNCTION:** Output Compare

[I/O]

**DESCRIPTION:**

The OC command allows the generation of output pulses based on the main encoder positions. The output is a low-going pulse with a duration of approximately 600 nanoseconds and is available at the output compare signal.

The auxiliary encoder cannot be used while using this function.

**Note: The OC function requires that the main encoder and auxiliary encoders be configured exactly the same (see the command, CE). For example: CE 0, CE 10.**

The output on pin 7 of the 5 CN connector is a TTL signal and requires JP3 to be installed.

**ARGUMENTS:** *OCX = m, n where*

m = Absolute position for first pulse. Integer between  $-2 \cdot 10^9$  and  $2 \cdot 10^9$

n = Incremental distance between pulses. Integer between -65535 and 65535.

OCx = 0 will disable the Output Compare function.

The sign of the parameter, n, will designate the expected direction of motion for the output compare function. When moving in the opposite direction, output compare pulses will occur at the incremental distance of  $65536 - |n|$  where  $|n|$  is the absolute value of n.

**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:**

\_OCx contains the state of the OC function

\_OCx = 0 : OC function has been enabled but not generated any pulses.

\_OCx = 1: OC function not enabled or has generated the first output pulse.

**EXAMPLES:**

OCX=300,100	Select X encoder as position sensor. First pulse at 300. Following pulses at 400, 500...
-------------	--

# OE

**FUNCTION:** 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, e, f, g, h where*

*x, y z, w, or a, b, c, d, e, f, g, h are unsigned integers*

*“?” returns the state of the off-on-error function*

**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

**FUNCTION:** 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.

**ARGUMENTS:** *OF x, y, z, w or OFX=x or OF a, b, c, d, e, f, g, h where*

x, y z, w, or a, b, c, d, e, f, g, h are signed integers

“?” returns the offset

**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

**FUNCTION:** Output Port

[I/O]

**DESCRIPTION:**

The OP command sets 4 bits of data on the output port of the controller simultaneously.

The n parameter is used to specify the number of bits affected starting with the LSB. The other bits are masked. For example, if n=2, only outputs 1 and 2 will be changed by OP m. If the n parameter is not specified, all bits will be changed.

To set or read outputs on a slave controller use the SA command.

**ARGUMENTS:** *OP m where*

m is an integer

**USAGE:**

While Moving	Yes	Minimum m Value	0
In a Program	Yes	Maximum m Value	15
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 output bit
"CB"	Clear output bit

**EXAMPLES:**

OP 0	Clear Output Port -- all bits
OP 3	Set outputs 1 and 2; clear the others
OP 7	Set outputs 1, 2 and 3.
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 all four outputs ON in slave controller on handle A

# @OUT

## FUNCTION: Status of Digital Output

### 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. A LEGEND-MC controller has 4 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:**

$$n = (\text{SlaveAddress} * 1000) + (\text{HandleNum} * 1000) + ((\text{Module}-1) * 4) + (\text{Bitnum}-1)$$

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	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

**FUNCTION:** Position Absolute

[Motion]

**DESCRIPTION:**

The PA command will set the absolute destination of the next move. The position is referenced to absolute zero. If a ? is used, then the current destination (current commanded position if not moving, destination if in a move) is returned. 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, e, f, g, h where

x, y, z, w, or a, b, c, d, e, f, g, h 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 contains current command position if not moving, 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
:PA ?	Returns the current commanded position
0000000	
:BG	Start the move
:PA 700	X-axis will go to 700 on the next move
:BG	

# PF

**FUNCTION:** 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 ?	PA ?
DE ?	PR ?
DP ?	TE
FL ?	
IP ?	
TP	

**ARGUMENTS:** *PF m.n* where

m is an integer. The negative sign for m specifies hexadecimal representation.

n is an integer

PF? Returns the value of m. n

**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

# PR

**FUNCTION:** 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. If a ? is used, then the current incremental distance is returned (even if it was set by a PA command). Units are in quadrature counts.

**ARGUMENTS:** *PR x, y, z, w or PRX=x or PR a, b, c, d, e, f, g, h where*

x, y z, w, or a, b, c, d, e, f, g, h are signed integers

“?” returns the current incremental distance

**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		

**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	

## QD

**FUNCTION:** Download Array**[General]****DESCRIPTION:**

The QD command transfers array data from the host computer to the LEGEND-MC. 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 array
------	--------------

# QR

**FUNCTION:** 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 the communication chapter of the user's manual. This command is not designed to be used in the application program, it is designed for data exchange with a computer.

**ARGUMENTS:** *QR xx where*

x is X,Y,Z,W,A,B,C,D,E,F,G,H or I or any combination to specify the axis, axes, or I/O status

I represents the status of the I/O

The Communication chapter of the users manual provides the definition of the data record information.

**USAGE:**

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

**RELATED COMMANDS:**

"QZ"	Return DMA / Data Record information
------	--------------------------------------

# QU

**FUNCTION:** Upload Array

[General]

**DESCRIPTION:**

The QU command transfers array data from the LEGEND-MC 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 array
------	----------------

# QW

**FUNCTION:** Slave Record Update Rate

[Motion]

**DESCRIPTION:**

The PR command is given to the master controller of a distributed system. The value establishes the update rate for data records to be sent from the slave controllers to the master controller. This command is executed on the master controller.

**ARGUMENTS:** *QWh=n where*

h is the handle being used to send commands to the slave controller.

n = an even integer between 4 and 16000. this sets the period at which the slave controller updates the master controller. the value of n represents the number of servo update cycles (default update cycle is 1 msec, see the TM command). The slave controller will always wait for this period after a data record has been sent before generating a new record.

**USAGE:**

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

**RELATED COMMANDS:**

"CH"	
"NA"	
"SR"	

**EXAMPLES:**

CHC=A,B	Using one LEGEND-MC as a master and one LEGEND-MC as a slave. This command assigns the slave, identified by the C axis designator, with Handle A for commands and Handle B for status returned from the slave.
QWB=20	Sets the update rate for the slave controller to 20 msec (TM=1000).

# QZ

**FUNCTION:** Return Data Record Information

[General]

**DESCRIPTION:**

The QZ command is an interrogation command that returns information regarding the Data Record. 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:**

"DR"	DMA update rate
"QR"	Data Record

# RA

**FUNCTION:** Record Array

[General]

**DESCRIPTION:**

The RA command selects up to four 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 [ ]      *where*

n,m 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 Interval

**EXAMPLES:**

#Record	Label
DM POS[100]	Define array
RA POS[]	Specify Record Mode
RD _TP	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

**FUNCTION:** Record

[General]

**DESCRIPTION:**

The RC command begins recording for the Automatic Record Array Mode (RA). RC 0 stops recording.

**ARGUMENTS:** RC n,m where

n is an integer 1 thru 8 and specifies 2<sup>n</sup> 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 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.

RC? returns status of recording. '1' if recording, '0' if not recording.

**USAGE:**

While Moving	Yes	Minimum n Value	1
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 Mode

**EXAMPLES:**

#RECORD	Record
DM Torque[1000]	Define Array
RA Torque[]	Specify Record Mode
RD _TT	Specify Data Type
RC 2	Begin recording, set 4 msec between records
JG 1000;BG	Begin motion
#A;JP #A,_RC=1	Loop until done
MG "DONE RECORDING"	Print message
EN	End program

# RD

FUNCTION: Record Data

[General]

**DESCRIPTION:**

The RD command specifies the data type to be captured for the Record Array (RA) 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* where

the arguments are the data type to be captured using the record array feature. The order is important. Each of the four 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 Interval
"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

**FUNCTION:** Return from Error Routine

[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 command to zero the subroutine stack. No RE is needed after ZS. After using ZS, use a JP command to return to a key location in the main program.

**ARGUMENTS:** RE n where

- 0 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

**FUNCTION:** Return from Interrupt Routine

[Program Flow]

**DESCRIPTION:**

The RI command is used to end the interrupt subroutine beginning with the label #ININT. An RI 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, RI1 restores the trippoint upon return to the program. RI0 clears a trippoint. To avoid returning after an interrupt, use the ZS 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 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"	Enable input interrupts

**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

**FUNCTION:** Report Latched Position

[General]

**DESCRIPTION:**

The RL command will return the last position captured by the latch. The latch must first be armed by the AL command. The armed state of the latch can be configured using the CN command.

**ARGUMENTS:** *RLn*      *where*

n = XYZW or ABCDEFGH for the main encoder latch and

n = SX, SY, SZ, SW or SA, SB, SC, SD, SE, SF, SG, SH for the auxiliary 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 or Sn contains the latched position where n is an axis letter.

**RELATED COMMAND:**

"AL"	Arm Latch
------	-----------

**EXAMPLES:**

JG 5000	Set up to jog
BG	Begin jog
AL	Arm the latch; assume that after about 2 seconds, input goes low
RL	Report the latch
10000	

## @RND

### FUNCTION: Round

**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

# RS

**FUNCTION:** 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:** *RSn*      *where*

0 (or no parameter) restores burned parameters and clears application program

1 restores burned parameters only

2 clears application programs only

**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

FUNCTION: Master Reset

[General]

**DESCRIPTION:**

The Master Reset command resets the LEGEND-MC to factory default settings and erases the EEPROM.

A master reset can also be performed by installing a jumper on the LEGEND-MC at the location labeled JP1/MR. The controller must be removed from the amplifier to access the jumper. The controller must be reattached to the amplifier and powered ON to perform the master reset. Remove the jumper after this procedure.

**USAGE:**

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

## <control>R<control>V

FUNCTION: 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

**FUNCTION:** Send Command

[General]

**DESCRIPTION:**

SA sends a command from the master to the slave controller of a distributed control system. Any command can be sent to a slave controller and will be interpreted by the slave as a "local" command. Some commands are only "local" commands and must be sent with the SA command. Refer to the discussion of local vs global commands in this manual.

**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" and the subsequent arguments as the arguments to the command: Example SAF= "KI",1,2 would send the command

**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 F 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 COMMAND:**

"CH"	Connect to Internet Handles for slaves
"IH"	Set Internet handles
"LO"	Lock out communication channels
"NA"	Set number of Axes for Distributed Control System
"QW"	Set Slave Data Record Update Rate

**EXAMPLES:**

CHY=A,B	Use one LEGEND-MC as a master and one LEGEND-MC as a slave. This command assigns the slave, identified by the Y axis designator, with Handle A for commands and Handle B for status returned from the slave.
SAA="KI",1,2	Sends the command to Handle A (slave controller): KI 1,2
SAA="TE"	Sends the command to Handle A (slave controller): TE
MG_SAA : 132	Display the content of the operand _SAA (first response to TE command)

# SB

**FUNCTION:** Set Bit

[I/O]

**DESCRIPTION:**

The SB command sets one of four bits on the output port, slave controller, or Modbus I/O.

**ARGUMENTS:** SB n where

n is an integer in the range 1 to 4 decimal or Modbus address. See chart below for setting outputs on slave controllers.

**DISTRIBUTED CONTROL:**

Handle	Command	Handle	Command
A	SB101 ~ SB104	E	SB501 ~ SB504
B	SB201 ~ SB204	F	SB601 ~ SB604
C	SB301 ~ SB304	G	SB701 ~ SB704
D	SB401 ~ SB404	H	SB801 ~ SB804

**MODBUS:**

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

$$n = (\text{SlaveAddress} * 1000) + (\text{HandleNum} * 1000) + ((\text{Module}-1) * 4) + (\text{Bitnum}-1)$$

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
------	-----------

**EXAMPLES:**

SB 3	Set output line 3
SB 1	Set output line 1
SB 602	Set output 2 on slave controller on handle F

# SC

**FUNCTION:** 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	9	Stopped after Finding Edge (FE)
1	Motors stopped at commanded independent position	10	Stopped after Homing (HM)
2	Decelerating or stopped by FWD limit switch or software limit, FL	11	Stopped by selective Abort Input
3	Decelerating or stopped by REV limit switch or software limit, BL	50	Contour running
4	Decelerating or stopped by Stop Command (ST)	51	Contour Stop
6	Stopped by Abort input	99	MC timeout
7	Stopped by Abort command (AB)	100	Motors are running, vector sequence
8	Decelerating or stopped by Off-on-Error (OE1)	101	Motors stopped at commanded vector

**ARGUMENTS:** SC XYZW or ABCDEFGH

**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		

**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

FUNCTION: 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.

This command can be useful when the position of a motor has been manually adjusted following a motor off (MO) command.

The SH command is integrated with the RUN output of the LEGEND amplifier. If the RUN output does not come ON within 100 msec, the controller returns to MO status and issues a command error. If this occurs, check the power on L1, L2, and L3.

**ARGUMENTS: SH XYZW or ABCDEFGH**

**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

### FUNCTION: Sin

#### 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

# SP

FUNCTION: 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.

**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, e, f, g, h* where

x, y, z, w, or a, b, c, d, e, f, g, h are unsigned integers

“?” returns the speed

**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		

**OPERAND USAGE:**

\_SPn contains the current speed setting where n is an axis letter.

**RELATED COMMANDS:**

"AC"	Acceleration
"DC"	Deceleration
"PR"	Position Relation
"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

### FUNCTION: Square Root

#### 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 @SIN[VAR1]	Display the value of the sine of VAR1
VAR2=@SIN[VAR1]+9	Perform calculation
EN	End of program

# ST

**FUNCTION:** Stop

[Motion]

**DESCRIPTION:**

The ST command stops commanded motion. The motor will come to a decelerated stop.

**ARGUMENTS:** *ST XYZWS or ABCDEFGH where*

XYZW or ABCDEFGH are axis designators. S indicates an interpolation sequence. 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		
Can be Interrogated	No		
Used as an Operand	No		

**RELATED COMMANDS:**

"BG"	Begin Motion
"MC"	Wait for motion to complete
"DC"	Deceleration rate

**EXAMPLES:**

ST	Stop motion
----	-------------

**HINT:** Use the *after motion complete command, AM, to wait for motion to be stopped.*

# TB

**FUNCTION:** 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	Controller addressed
Bit 6	Executing program
Bit 5	Contouring
Bit 4	Executing error or limit switch routine
Bit 3	Input interrupt enabled
Bit 2	Executing input interrupt routine
Bit 1	0 (Reserved)
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

FUNCTION: Tell Error 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
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

91	Amplifier not in run status
97	Bad binary command format
98	Binary Commands not valid in application program
99	Bad binary command number
100	Not valid when running ECAM
101	Improper index into ET (must be 0-256)
102	No master axis defined for ECAM
103	Master axis modulus greater than 256*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

**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

**FUNCTION:** Tell Dual Encoder

[Motion]

**DESCRIPTION:**

This command returns the current position of the dual (auxiliary) encoder.

**ARGUMENTS:** *TD XYZW or ABCDEFGH*

**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 Encoder
------	--------------

**EXAMPLES:**

:PF 7	Position format of 7
:TD	Return Dual encoder
0000200	
DUAL=_TDX	Assign the variable, DUAL, the value of TD

# TE

FUNCTION: Tell Error

[Status]

**DESCRIPTION:**

This command returns the current position error of the motor. It is up-dated every servo cycle.

**ARGUMENTS: TE XYZW or ABCDEFGH**

**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:**

\_TE<sub>n</sub> 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.

# TI

**FUNCTION:** 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	19
Bit 5	Input 6	42
Bit 4	Input 5	43
Bit 3	Input 4	44
Bit 2	Input 3	45
Bit 1	Input 2	17
Bit 0	Input 1	18

**DISTRIBUTED CONTROL:**

Handle	Command	Handle	Command
A	TI 100	E	TI 500
B	TI 200	F	TI 600
C	TI 300	G	TI 700
D	TI 400	H	TI 800

**ARGUMENTS:** TI

**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\*

**FUNCTION:** Time Operand (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 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

# TL

**FUNCTION:** 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.

**ARGUMENTS:** *TL x, y, z, w or TLX=x or TL a, b, c, d, e, f, g, h where*

x, y z, w, or a, b, c, d, e, f, g, h are unsigned integers

“?” returns the limit value

**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		

**OPERAND USAGE:**

\_TLn contains the value of the torque limit where n is an axis letter.

**EXAMPLES:**

TL 1	Limit X-axis torque to 1 volt
TL ?	Return torque limit
1.0000	

# TM

FUNCTION: Time

[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  $\mu\text{sec}$ . If a multi-axis system is configured the TM value is set in all controllers if set in the master.

**ARGUMENTS:** *TM n*     *where*

n is an integer in microseconds with a resolution of 125 microseconds.

“?” returns the value of the sample clock

**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 $\mu\text{sec}$ (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 command to SP commands.**

# TP

**FUNCTION:** Tell Position

[Status]

**DESCRIPTION:**

This command returns the current position of the motor in quadrature counts. This value is up-dated every servo cycle.

**ARGUMENTS:** *TP XYZW or ABCDEFGH*

**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:**

\_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

FUNCTION: Trace

[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

**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

FUNCTION: 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 CN 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:** TS XYZW or ABCDEFGH

**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

**FUNCTION:** 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 up-dated every servo cycle.

**ARGUMENTS:** *TT XYZW or ABCDEFGH*

**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	No	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
------	--------------

**EXAMPLES:**

V1=_TT	Assigns value of TT to variable, V1
TT	Report torque
-0.2843	Torque is -.2843 volts

# TV

**FUNCTION:** 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 ABCDEFGH

**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	No	Default Format	8.0
Used as an Operand	Yes		

**OPERAND USAGE:**

\_TVn contains the value for the velocity where n is an axis letter.

**EXAMPLES:**

VELX=_TV	Assigns value of velocity to the variable VELX
TV	Returns the velocity
0003420	

# TW

**FUNCTION:** Timeout for IN-Position (MC)

[Setting]

**DESCRIPTION:**

The TW n command sets the timeout in msec to declare an error if the MC 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 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, e, f, g, h where

x, y z, w, or a, b, c, d, e, f, g, h are signed integers  
 n specifies timeout in msec, -1 disables the timeout  
 “?” returns the timeout in msec for the MC command

**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		

**OPERAND USAGE:**

\_TW contains the timeout in msec for the MC command .

**RELATED COMMANDS:**

"MC"	Motion Complete - "In Position"
------	---------------------------------

# UL

**FUNCTION:** Upload**[General]****DESCRIPTION:**

The UL command transfers data from the LEGEND-MC 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 126.

**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

FUNCTION: Vector Acceleration

[Motion]

**DESCRIPTION:**

This command sets the acceleration rate of the vector in a coordinated motion sequence.

**ARGUMENTS:** VA n *where*

n is an unsigned integer. The parameter input will be rounded down to the nearest factor of 1024. The units of the parameter is counts per second squared.

n = ? Returns the value of the vector acceleration for the coordinate plane.

**USAGE:**

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

**OPERAND USAGE:**

\_VA contains the value of the vector acceleration.

**RELATED COMMANDS:**

"VS"	Vector Speed
"VP"	Vector Position
"VE"	End Vector
"VM"	Vector Mode
"BGS"	Begin Sequence
"VD"	Vector Deceleration
"VS"	Vector smoothing constant - S-curve

**EXAMPLES:**

VA 1024	Set vector acceleration to 1024 counts/sec <sup>2</sup>
VA ?	Return vector acceleration
00001024	
VA 20000	Set vector acceleration
VA ?	
0019456	Return vector acceleration
ACCEL=_VA	Assign variable, ACCEL, the value of VA

# VD

**FUNCTION:** Vector Deceleration

[Motion]

**DESCRIPTION:**

This command sets the deceleration rate of the vector in a coordinated motion sequence.

**ARGUMENTS:** *VD n where*

n is an unsigned integer. The parameter input will be rounded down to the nearest factor of 1024. The units of the parameter is counts per second squared.

n = ? Returns the value of the vector deceleration for the coordinate plane.

**USAGE:**

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

**OPERAND USAGE:**

\_VD contains the value of the vector deceleration.

**RELATED COMMANDS:**

"VA"	Vector Acceleration
"VS"	Vector Speed
"VP"	Vector Position
"VE"	Vector End
"VM"	Vector Mode
"BGS"	Begin Sequence
"VT"	Smoothing constant - S-curve

**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

FUNCTION: Vector Sequence End

[Motion]

**DESCRIPTION:**

VE is required to specify the end segment of a coordinated move sequence. VE follows the final VP or CR command in a sequence. VE is equivalent to the LE command. If a VE command is not issued before the controller runs all the linear segments, motion will stop instantaneously.

**ARGUMENTS: VE**

n = ? Returns the length of the vector in counts.

**USAGE:**

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

**OPERAND USAGE:**

\_VE contains the length of the vector in counts.

**RELATED COMMANDS:**

"VM"	Vector Mode
"VS"	Vector Speed
"VA"	Vector Acceleration
"VD"	Vector Deceleration
"VP"	Vector Position
"BGS"	Begin Sequence
"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

## FUNCTION: 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.

VF? returns the value of the format for variables and arrays.

### 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

# VR

## FUNCTION: 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 or the operators < and > used with CR, VP and LI 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 and VD.

### ARGUMENTS: VR n where

n is an integer with a resolution of .0001.

n = ? Returns the value of the vector speed ratio.

### 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:

\_VR contains the vector speed ratio.

### RELATED COMMANDS:

"VS" on page	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@AN[1]*.1	Read analog input compute ratio
JP#SPEED	Loop
XQ#A,0; XQ#SPEED,1	Execute task 0 and 1 simultaneously

**Note: Use VR for feedrate override, when specifying the speed of individual segments using the operator '<'.**

# VS

**FUNCTION:** Vector Speed

[Motion]

**DESCRIPTION:**

The VS command specifies the speed of the vector in a coordinated motion sequence in either the LM or VM 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 n where

n is an unsigned even number. The units are counts per second.

n = ? Returns the value of the vector speed.

**USAGE:**

While Moving	Yes	Minimum n Value	2
In a Program	Yes	Maximum n Value	12,000,000
Command Line	Yes	Default Value	25000
		Default Format	---

**OPERAND USAGE:**

\_VS contains the vector speed.

**RELATED COMMANDS:**

"VA"	Vector Acceleration
"VP"	Vector Position
"LM"	Linear Interpolation
"VM"	Vector Mode
"BGS"	Begin Sequence
"VE"	Vector End

**EXAMPLES:**

VS 2000	Define vector speed
VS ?	Return vector speed
002000	

*Hint: Vector speed can be attached to individual vector segments. For more information, see description of VP, CR, and LI commands.*

# VT

**FUNCTION:** Vector Time Constant - S curve

[Motion]

**DESCRIPTION:**

The VT command filters the acceleration and deceleration functions in vector moves of VM, LM 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 n* *where*

n is an unsigned number with a resolution of 1/256.

n = ? Returns the value of the vector time constant.

**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
		Default Format	1.4

**OPERAND USAGE:**

\_VT contains the vector time constant.

**RELATED COMMANDS:**

"IT"	Independent Time Constant for smoothing independent moves
------	---

**EXAMPLES:**

VT 0.8	Set vector time constant
VT ?	Return vector time constant
0.8	

# WC

**FUNCTION:** Wait for Contour Data

[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"	Contour 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

# WT

FUNCTION: 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 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 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

**FUNCTION:** 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. Two programs may be executed simultaneously to perform multitasking.

**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 or 1

**NOTE:** The arguments for the command, XQ, are optional. If no arguments are given, the first program in memory will be executed as thread 0.

**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:**

\_XQ contains the current line number of execution for thread n, and -1 if thread n is not running.

**RELATED COMMANDS:**

"HX" on page	Halt execution
--------------	----------------

**EXAMPLES:**

XQ #Apple,0	Start execution at label Apple, thread zero
XQ #data,1	Start execution at label data, thread one
XQ 0	Start execution at line 0

# ZA

**FUNCTION:** User Variable

[Configuration]

**DESCRIPTION:**

ZA sets the first user variable for use with a distributed control system. The user variable is automatically sent as part of the status record from the slave controller to the master controller. This variable provides a method for specific slave information to be passed to the master automatically.

**ARGUMENTS:** *ZA n, n, n, n, n, n, n or ZAA=n where*

n can be a number, controller operand, variable, mathematical function, or string. The range for numeric values is 4 bytes of integer 2147483647 ( $2^{31}$ ). The maximum number of characters for a string is 4 characters. Strings are identified by quotations.

**USAGE:**

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

**OPERAND USAGE:**

ZAa contains the user variable defined by the axis, a, set with a ZA command. a is any axis designator A, B, C, D, E, F, G, and H.

**RELATED COMMANDS:**

"ZB"	Set second variable
------	---------------------

**EXAMPLES:**

CHB=A, B	Use one LEGEND-MC as a master and one LEGEND-MC as a slave. This command assigns the slave, identified by the S axis designator, with handle A for commands and handle B for status returned from the slave.
QWB=20	Sets the update rate for the slave controller to 20msec (TM=1000).
ZA 2343.43	Sets the first user variable to a number (2343).

# ZB

**FUNCTION:** User Variable, ZB

[Configuration]

**DESCRIPTION:**

ZB sets the “B” user variable for use with a distributed control system. The user variable is automatically sent as part of the status record from the slave controller to the master controller. This variable provides a method for specific slave information to be passed to the master automatically.

**ARGUMENTS:** *ZB n, n, n, n, n, n, n or ZBA=n where*

n can be a number, controller operand, variable, mathematical function, or string. The range for numeric values is 4 bytes of integer 2147483647 ( $2^{31}$ ). The maximum number of characters for a string is 4 characters. Strings are identified by quotations.

**USAGE:**

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

**OPERAND USAGE:**

ZBa contains the user variable defined by the axis, a, set with a ZB command. a is any axis designator A, B, C, D, .E ,F ,G, and H.

**RELATED COMMANDS:**

"ZA"	Set first variable
------	--------------------

**EXAMPLES:**

CHB=A, B	Use one LEGEND-MC as a master and one LEGEND-MC as a slave. This command assigns the slave, identified by the S axis designator, with handle A for commands and handle B for status returned from the slave.
QWB=20	Sets the update rate for the slave controller to 20msec (TM=1000).
ZB "DOG"	Sets the first user variable to the string "DOG".

# ZS

**FUNCTION:** 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. This turns the jump to subroutine into a jump.

**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	16
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 or 1. The response, an integer between zero and seven, 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

COMMAND INTERROGATION LIST					
Command	Definition	Units	Minimum	Maximum	Default
_AB	Status of abort input	status	0=Aborted	1=OK	n/a
_ACx	Axis acceleration rate	counts/sec <sup>2</sup>	1024	67107840	256000
_AFx	Analog or digital feedback?	status	0=DIGITAL	1=ANALOG	0
_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
_BLx	Reverse software limit	counts	-2147483648	2147483647	-2147483648
_BN	Serial number of the SMC3010	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
_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	14	14
_DCx	Axis deceleration rate	counts/sec <sup>2</sup>	1024	67107840	256000
_DEx	Encoder position of the auxiliary encoder	counts	-2147483648	2147483647	n/a
_DL	Number of available labels	quantity	0	126	126
_DM	Number of available array locations	quantity	0	8000	8000
_DPx	Current encoder position of axis	counts	-2147483648	2147483647	n/a
_DT	Time interval for contour mode	2 <sup>N</sup> 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	359	0
_ED	The last line that caused a CMDERR	line number	0	499	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
_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

Command	Definition	Units	Minimum	Maximum	Default
_HXx	Thread info	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	0
_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
_Klx	Integrator for PID loop	constant	0	2047.875	0
_KPX	Proportional Constant for PID loop	constant	0	1023.875	6
_LE	Length of the vector	counts	0	2147483647	0
_LFx	Forward Limit Switch	status	0=ACTIVE	1=INACTIVE	n/a
_LM	Number of free segments 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	499	0
_LZ	Serial port leading zero removal	status	0=OFF	1=ON	0
_MOx	Current state of motor, enabled or not	status	0=ENABLED	1=DISABLED	0 / 1
_MTx	Type of motor	configuration	-1	1	1
_NA	Returns total number of axes (distributed)	configuration	1	8	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
_OCx	Returns the state of the Output Compare Function	status	0	1	n/a
_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
_PAx	Last commanded absolute position if moving, otherwise current position	counts	-2147483648	2147483647	0

Command	Definition	Units	Minimum	Maximum	Default
_PRx	Current incremental distance to move (Even if move set by PA)	counts	-2147483648	2147483647	0
_RC	Status of record mode	status	0= NOT RECORDING	1=RECORDING	0= NOT RECORDING
_RD	next array element to be recorded	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
_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
_Tx	8 inputs as a decimal or hex value (x = input bank 0-7)	byte	0	255	n/a
TIME	Counter since controller was reset	servo cycles	0	2147483647	0
_TLx	Torque limit of axis	voltage	0	9.9988	9.9988
_TM	Servo update cycle for all axes	uSec	375	20000	1000
_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
_UL	Number of variables available	n/a	0	254	254
_VA	acceleration value for vector mode	counts/sec <sup>2</sup>	1024	68431360	256000
_VD	Deceleration value for vector mode	counts/sec <sup>2</sup>	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	499	n/a
_ZS	Current subroutine depth	number	0	16	n/a



To view the current values for each command, specify the command followed by a ? for each axis requested. The LEGEND-MC 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,D,E,F,G or H for the LEGEND-MC ). 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 X,Y,Z,W values
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 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 ABCDEFGH	Begin all axes
BG D	Begin D only

## Controller Response to Commands

For each valid command entered, the LEGEND-MC returns a colon (:). If the LEGEND-MC decodes a command as invalid, it returns a question mark (?).

**Note: The LEGEND-MC returns a : for valid commands.**

**Note: The LEGEND-MC returns a ? for invalid commands.**

For example, if the command bg is sent in lower case, the LEGEND-MC will return a ?.

:bg <enter>	Invalid command (lower case)
?	LEGEND-MC returns a ?

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 Tell Position (TP) or Tell Status (TS), the LEGEND-MC 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	000000000
Tell Position X and Y	:TP XY <enter>
data returned	000000000,000000000

The format of the returned data can be set using the Position Format (PF) and Variable Format (VF) command.

:PF 4 <enter>	Position Format is 4 integers
:TP X <enter>	Tell Position
0000	returned data

# Command Summary

Each LEGEND-MC 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, KD, and KI 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
LS	List
MO	Motor Off
MT	Motor Type
OB	Output Bit
OP	Output Port
PF	Position Format
QU	Upload Array
QD	Download 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
OF	Offset
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
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

## 6 Programming Motion

### Overview

The LEGEND-MC 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 LEGEND-MC 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.

<u>Example Application</u>	<u>Mode of Motion</u>	<u>Commands</u>
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
1 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
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 LEGEND-MC 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 sent by the LEGEND-MC profiler.

**Note: The actual motor motion may not be complete when the profile has been completed, however, the next motion command may be specified.**

The Begin (BG) 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 speed (SP) and the acceleration (AC) can be changed at any time during motion, however, the deceleration (DC) and position (PR or PA) cannot be changed until motion is complete. Remember, motion is complete (AM) when the profiler is finished, not when the actual motor is in position. The Stop command (ST) can be issued at any time to decelerate the motor to a stop before it reaches its final position.

An incremental position movement (IP) 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 and BG command combination.

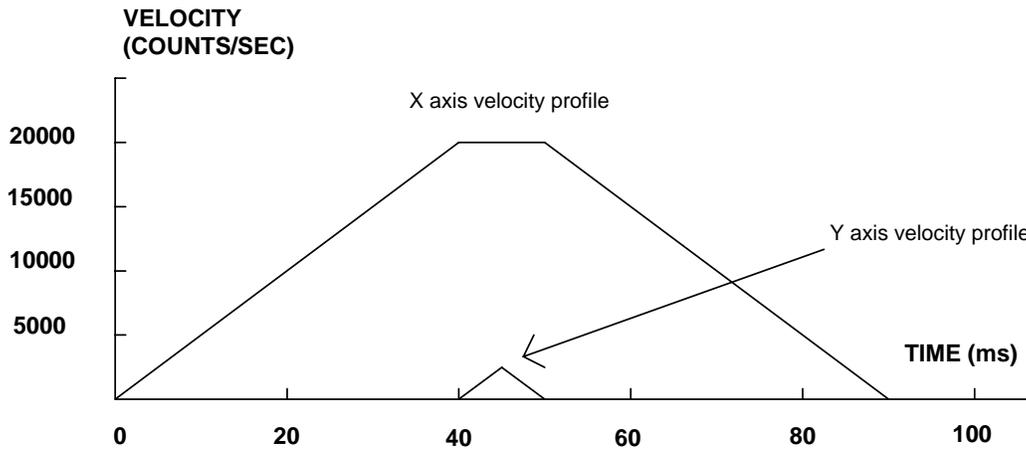
### Command Summary - Independent Axis

<u>Command</u>	<u>Description</u>
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 SC3010 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.

<b>Instruction</b>	<b>Interpretation</b>
#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 <sup>2</sup> for all axes
DC 500000,500000	Specify deceleration of 500000 counts / sec <sup>2</sup> 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.

## Independent Jogging

The jog mode of motion is very flexible because speed, direction and acceleration can be changed during motion. The user specifies the jog speed (JG), acceleration (AC), and the deceleration (DC) rate for each axis. The direction of motion is specified by the sign of the JG parameters. When the begin command is given (BG), the motor accelerates up to speed and continues to jog at that speed until a new speed or stop (ST) 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 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 LEGEND-MC 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

<b>Command</b>	<b>Description</b>
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 LEGEND-MC provides a linear interpolation mode for 1 axis. 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 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 Linear Segments

The command LI 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 clear sequence (CS) 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, STT, or AB. The command, ST, causes a decelerated stop. The command, AB, causes an instantaneous stop and aborts the program, and the command AB1 aborts the motion only.

The Linear End (LE) 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 LEGEND-MC 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 n, VA n, and VD n are used to specify the vector speed, acceleration, and deceleration. The LEGEND-MC computes the vector speed based on the axes specified in the LM mode. For example, LM XY designates linear interpolation for the X and Y axes. The vector speed for this example would be computed using the equation:

$VS^2=XS^2+YS^2$ , where XS and YS are the speed of the X and Y axes.

The controller computes the vector speed with the axis specifications from LM.

VT is used to set the S-curve smoothing constant for coordinated moves. The command AV 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 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,y < n > m

The first command, < n, is equivalent to commanding VS<sub>n</sub> 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 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.

As an example, consider the following program.

<b><u>Instruction</u></b>	<b><u>Interpretation</u></b>
#ALT	Label for alternative program
DP 0,0	Define Position of X and Y axis to be 0
LMXY	Define linear mode between X and Y axes.
LI 4000,0 <4000 >1000	Specify first linear segment with a vector speed of 4000 and end speed 1000
LI 1000,1000 < 4000 >1000	Specify second linear segment with a vector speed of 4000 and end speed 1000
LI 0,5000 < 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 .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

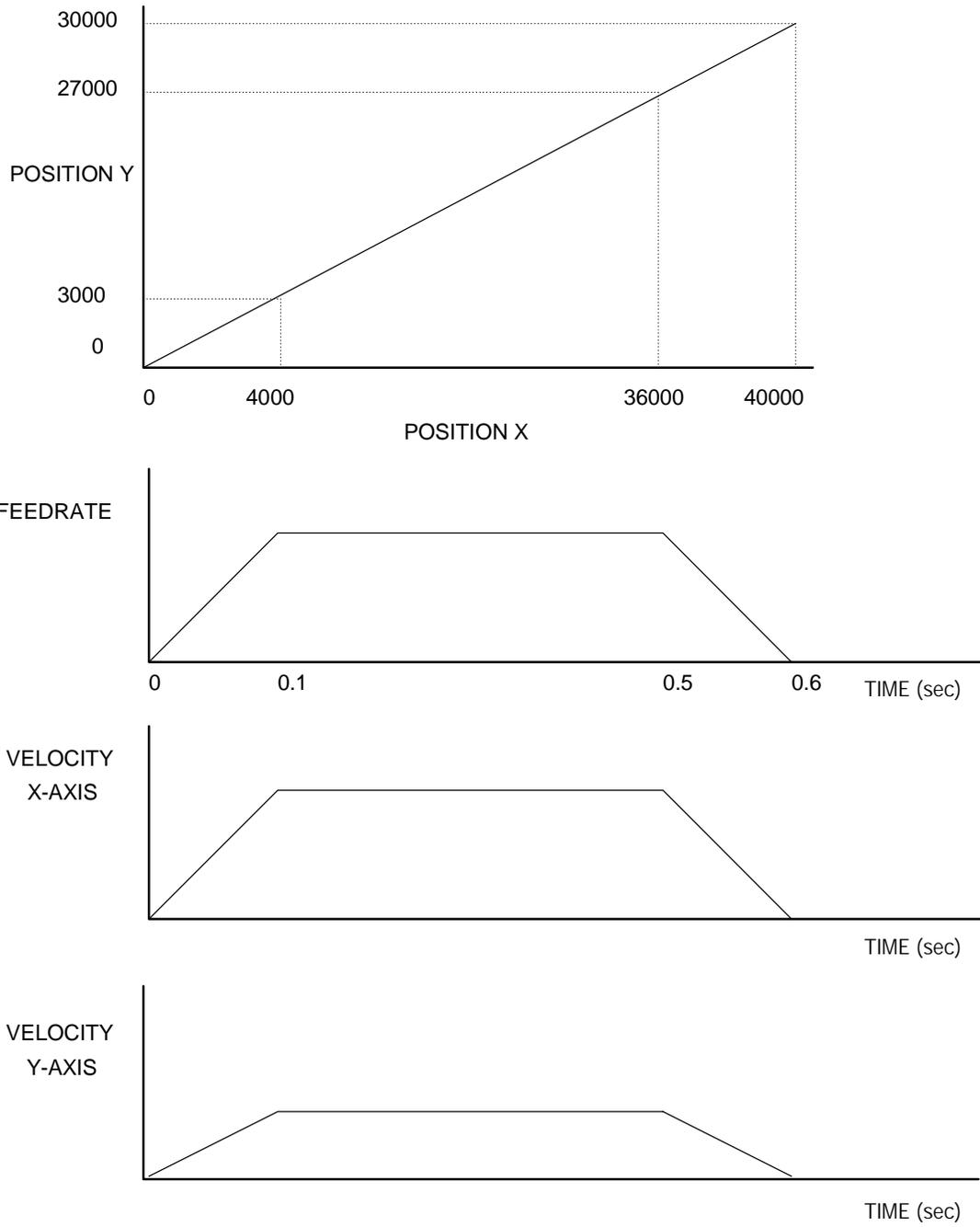
<b>Command</b>	<b>Description</b>
LM xy	Specify axes for linear interpolation
LM?	Returns number of available spaces for linear segments in LEGEND-MC sequence buffer. Zero means buffer full. 511 means buffer empty.
LI x,y < n	Specify incremental distances relative to current position, and assign vector 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
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, \_VPX and \_VPY will be zero.

Now suppose that the interrogation is repeated at the second segment when Y=2000. The value of \_AV at this point is 7000, \_CS equals 1, \_VPX=5000 and \_VPY=0.

Note that the above program specifies the vector speed, VS, and not the actual axis speeds VX and VY. The axis speeds are determined by the LEGEND-MC.

The resulting profile is shown in the following illustration - *Linear Interpolation*.



*Linear Interpolation*

## Vector Mode: Linear Interpolation Motion

### Specifying Vector Segments

The motion segment is described by the command; VP for linear segments. Once a set of linear 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 x,y specifies the coordinates of the end points of the vector movement with respect to the starting point.

Up to 511 VP segments 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 commands which were stored in the buffer prior to the start of the motion. To stop the motion, use the instructions STS or AB1. ST 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 LEGEND-MC 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 speed.

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,y < n > m

The first parameter, <n, is equivalent to commanding VS<sub>n</sub> at the start of the given segment and will cause an acceleration toward the new commanded speeds, subjects 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 and 10 times with a resolution of .0001. This command takes effect immediately and causes VS 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 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.

## Command Summary - Coordinated Motion Sequence

<b><u>Command</u></b>	<b><u>Description</u></b>
VM m,n	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.
VT	S curve smoothing constant for coordinated moves
LM?	Return number of available spaces for linear and circular segments in LEGEND-MC sequence buffer. Zero means buffer is full. 512 means buffer is empty.

## Operand Summary - Coordinated Motion Sequence

<b><u>Operand</u></b>	<b><u>Description</u></b>
_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 LEGEND-MC 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.

## Electronic Gearing

With the LEGEND-MC, the master is always the auxiliary encoder. The master may rotate in both directions and the geared axis will follow at the specified gear ratio.

The GA command is unnecessary for the LEGEND-MC, as the auxiliary encoder is automatically used. GR x,y specifies the gear ratios for the slaves where the ratio may be a number between +/-127.9999 with a fractional resolution of .0001. GR 0,0 turns off gearing in both modes. A limit switch or ST command disables gearing.

GR causes the specified axes to be geared to the actual position of the master.

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, JG, VP, or LI commands.

### Command Summary - Electronic Gearing

<b><u>Command</u></b>	<b><u>Description</u></b>
GA n	Specifies master axes for gearing where n=DX for auxiliary encoder.
GR x	Sets gear ratio for slave axes. 0 disables electronic gearing .
GR a	Sets gear ratio for slave axes. 0 disables electronic gearing.
MR x	Trippoint for reverse motion past specified value.
MF x	Trippoint for forward motion past specified value.

## Electronic Cam

The electronic cam is a motion control mode which enables the periodic synchronization of the servo motor with an external device. The LEGEND-MC uses the auxiliary encoder as the master axis.

The electronic cam is a more detailed type of electronic gearing which allows a table-based relationship between the axes. To illustrate the procedure of setting the cam mode, consider the cam relationship for the slave axis X. 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

EAD is the auxiliary encoder for the x-axis

For the given example, since the master is x, we specify EADX

### 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, we use the instruction EM and MM.

EM x; MMx

where EMx specifies the cycle of the slave over one cycle and MMx specifies the cycle of the master.

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; MM 6000

### 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 256 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 256. 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.

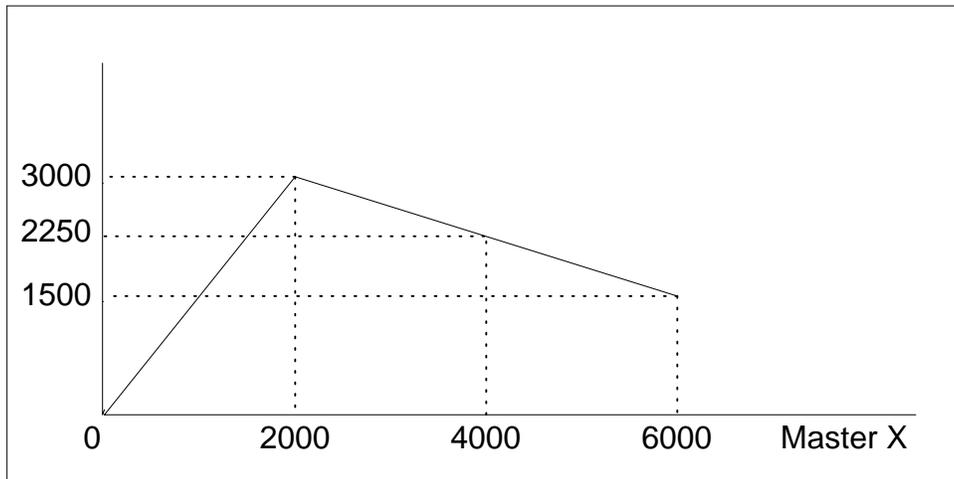
If the value of any parameter is outside the range of one cycle, the cam engages immediately. When the cam is engaged, the slave position is redefined, modulo one cycle.

**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.

<u>Instruction</u>	<u>Interpretation</u>
#SETUP	Label
EAX	Select X as master
EM 1000	Specify slave cycle
EP 20,0	Master position increments
MM 1000	Specify master cycle
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:

<b><u>Instruction</u></b>	<b><u>Interpretation</u></b>
#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

## Contour Mode

The LEGEND-MC also provides a contouring mode. This mode allows any arbitrary position curve to be prescribed for any motion 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

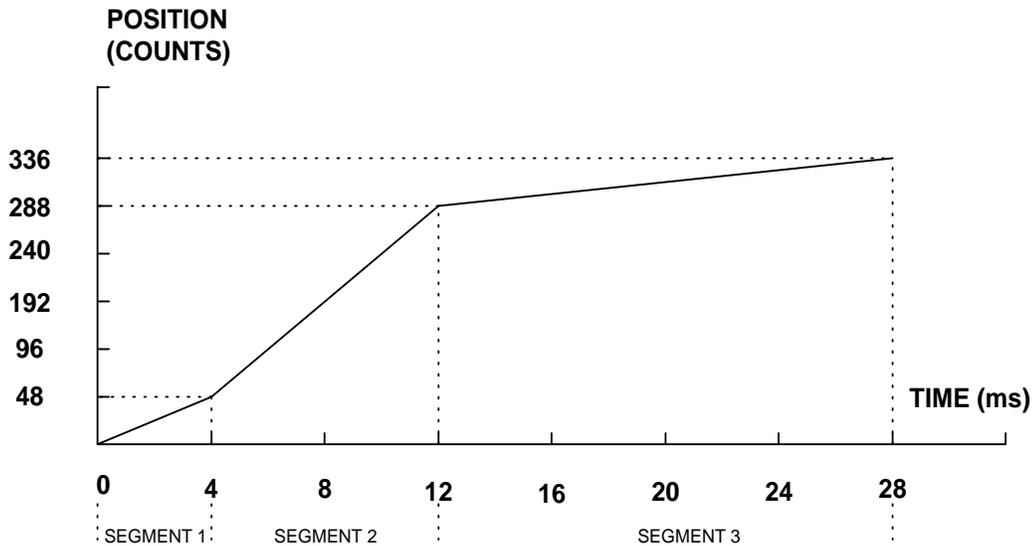
The same trajectory may be represented by the increments

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:

<b>Instruction</b>	<b>Description</b>
#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 LEGEND-MC 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

<b>Command</b>	<b>Description</b>
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 LEGEND-MC 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 and VT 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 S curve, has continuous acceleration and results in reduced mechanical vibrations.

The smoothing function is specified by the following commands:

<b>Command</b>	<b>Description</b>
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.

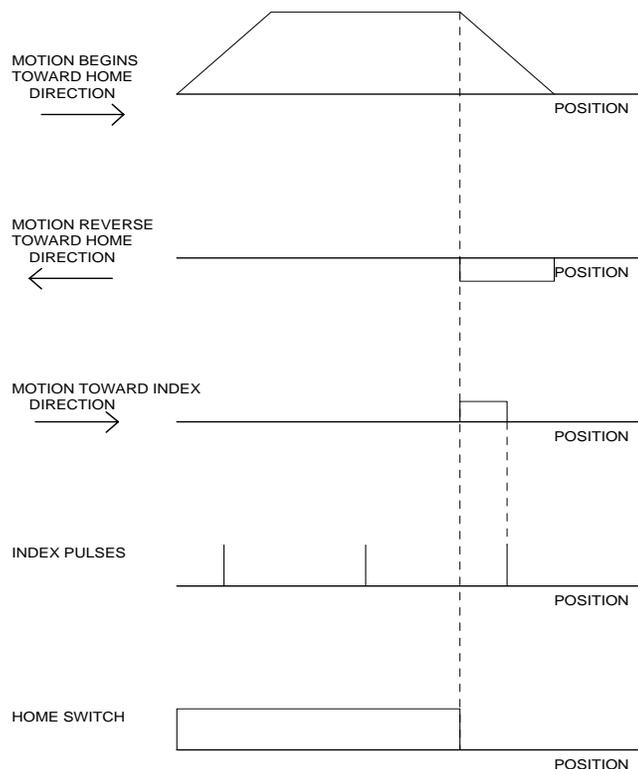
## Homing

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; +5V 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 LEGEND-MC 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 LEGEND-MC 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 encoder.

**Note:** To insure a position capture within 25 microseconds, the input signal must be a transition from high to low.

The LEGEND-MC 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 (LEGEND) 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.

To capture the position of the auxiliary encoder, use the command ALSX. The input must be wired to general input 2. \_RLSX holds the captured position.

# 7 Application Programming

## Introduction

The LEGEND-MC programming language is a powerful language that allows users to customize a program to handle their application. Complex programs can be downloaded into the LEGEND-MC memory for later execution. Utilizing the LEGEND-MC 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 LEGEND-MC provides commands that allow the LEGEND-MC 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 LEGEND-MC provides 254 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

A LEGEND-MC program consists of several LEGEND-MC 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 LEGEND-MC 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 LEGEND-MC 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
#CMDERR	Label for incorrect command subroutine
#COMINT	Label for communication interrupt
#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 and 1. 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 and HX 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.

## Debugging Programs

The LEGEND-MC 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 LEGEND-MC 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. This command reports back a number and error condition as follows:

### Error Codes:

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 while 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 program/strand already executed
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
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
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
59	Mismatched parentheses
60	Download error - line too long or too many lines
61	Duplicate or bad label
62	Too many labels
63	IF command 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
91	Amplifier not in run status
97	Bad binary command format
98	Binary commands not valid in application format
99	Bad binary command number
100	Not valid when running ECAM
101	Improper index to ET (must be 0-256)
102	No master axis for ECAM
103	Master axis modulus greater than 256* 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

## Program Flow Commands

The LEGEND-MC provides instructions that control program flow. The LEGEND-MC 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 LEGEND-MC 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 LEGEND-MC 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 LEGEND-MC can make decisions based on its own status or external events without intervention from a host computer.

## LEGEND-MC 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.
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 LEGEND-MC 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 LEGEND-MC 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 LEGEND-MC to make decisions without a host computer. For example, the LEGEND-MC can decide between two motion profiles based on the state of an input line. Or, the LEGEND-MC 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 LEGEND-MC 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 LEGEND-MC executes operations from left to right.**

### ***Example using variables named V1, V2, V3 and V4:***

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 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

**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 LEGEND-MC 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 its 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.

### ***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 LEGEND-MC allows IF conditional statements to be included within other IF conditional statements. This technique is known as 'nesting' and the LEGEND-MC 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**

<b>Format:</b>	<b>Meaning</b>
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)	2 <sup>nd</sup> IF executed if 1 <sup>st</sup> IF conditional true
MG "INPUT 1 AND INPUT 2 ARE ACTIVE"	Message executed if 2 <sup>nd</sup> IF is true
ELSE	ELSE command for 2 <sup>nd</sup> IF statement
MG "ONLY INPUT 1 IS ACTIVE"	Message executed if 2 <sup>nd</sup> IF is false
ENDIF	End of 2 <sup>nd</sup> conditional statement
ELSE	ELSE command for 1 <sup>st</sup> IF statement
MG "ONLY INPUT 2 IS ACTIVE"	Message executed if 1 <sup>st</sup> IF statement
ENDIF	End of 1 <sup>st</sup> 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
RIO	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
-------	--------------------------------

## Automatic Subroutines for Monitoring Conditions

Often it is desirable to monitor certain conditions continuously without tying up the host or LEGEND-MC program sequences. The LEGEND-MC 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, predefined label in the applications program. The pre-defined labels are:

#CMDERR	Bad command given
#COMINT	Communication interrupt occurred
#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 LEGEND-MC 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 8 million), the #CMDERR routine will be executed prompting the operator to enter a new number.

## Mathematical and Functional Expressions

For manipulation of data, the LEGEND-MC 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 LEGEND-MC 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 LEGEND-MC 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.

### **Example:**

PR POSX	Assigns variable POSX to PR command
JG RPMY*70	Assigns variable RPMY multiplied by 70 to JG command.

### Programmable Variables

The LEGEND-MC allows the user to create up to 254 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 LEGEND-MC 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 LEGEND-MC 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

$$10 \text{ Volts} = 8191 \text{ counts} \rightarrow 3000 \text{ rpm} = 200000 \text{ c/sec}$$

$$\text{Speed/Analog input} = 200000/8191 = 24.4$$

#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 LEGEND-MC commands to be incorporated into programmable variables and expressions. Internal variables are designated by adding an underscore ( \_ ) prior to the LEGEND-MC command. LEGEND-MC commands which can be used as internal variables are listed in the Command Reference as "Used as an Operand".

Most LEGEND-MC 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 LEGEND-MC registers. The X,Y,Z or W or A,B,C,D,E,F,G,H for the LEGEND-MC, 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 LEGEND-MC also provides a few keywords which give access to internal variables that are not accessible by standard LEGEND-MC 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
INTIME=TIME	Initialize time variable
PR5000;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 LEGEND-MC provides array space for 8000 elements in up to 14 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 14 different arrays may be defined. The arrays are one dimensional.

#### **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 LEGEND-MC 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 LEGEND-MC.

**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 message command, MG. 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 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
000000010.0000	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 LEGEND-MC 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<sup>2</sup>. 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. ( $2^1 + 2^2 = 6$ )
OP0	Clears all bits of output port to zero
OP 255	Sets all bits of output port to one. ( $2^0 + 2^1 + 2^2 + 2^3 + 2^4 + 2^5 + 2^6 + 2^7$ )

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 LEGEND-MC 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

## Input Interrupt Function

The LEGEND-MC provides an input interrupt function which causes the program to automatically execute instructions following the #ININT label. This function is enabled using the II m,n,o command. m specifies the beginning input and n specifies the final input in the range. The parameter o is an interrupt mask. If m and n are unused, o contains a number with the mask. A 1 designates the input to be enabled for an interrupt, where  $2^0$  is bit 1,  $2^1$  is bit 2 and so on. For example, II,5 enables inputs 1 and 3 ( $2^0 + 2^2 = 5$ ).

A low input on any of the specified inputs will cause automatic execution of the #ININT subroutine. The Return from Interrupt (RI) command is used to return from this subroutine to the place in the program where the interrupt had occurred. If it is desired to return to somewhere else in the program after the execution of the #ININT subroutine, the Zero Stack (ZS) command is used followed by unconditional jump statements. See Section 10, Example Applications, for an example.

**Important:** Use the RI instruction (not EN) to return from the #ININT subroutine.

## Analog Inputs

The LEGEND-MC provides two analog inputs. The value of these inputs in volts may be read using the @AN[n] function where n is the analog input 1 or 2. The resolution of the Analog-to-Digital conversion is 14 bits. Analog inputs are useful for reading special sensors such as temperature, tension or pressure.

The examples in Section 10, Example Applications, show programs which cause the motor to follow an analog signal. The first example is a point-to-point move. The second shows a continuous move.

# 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

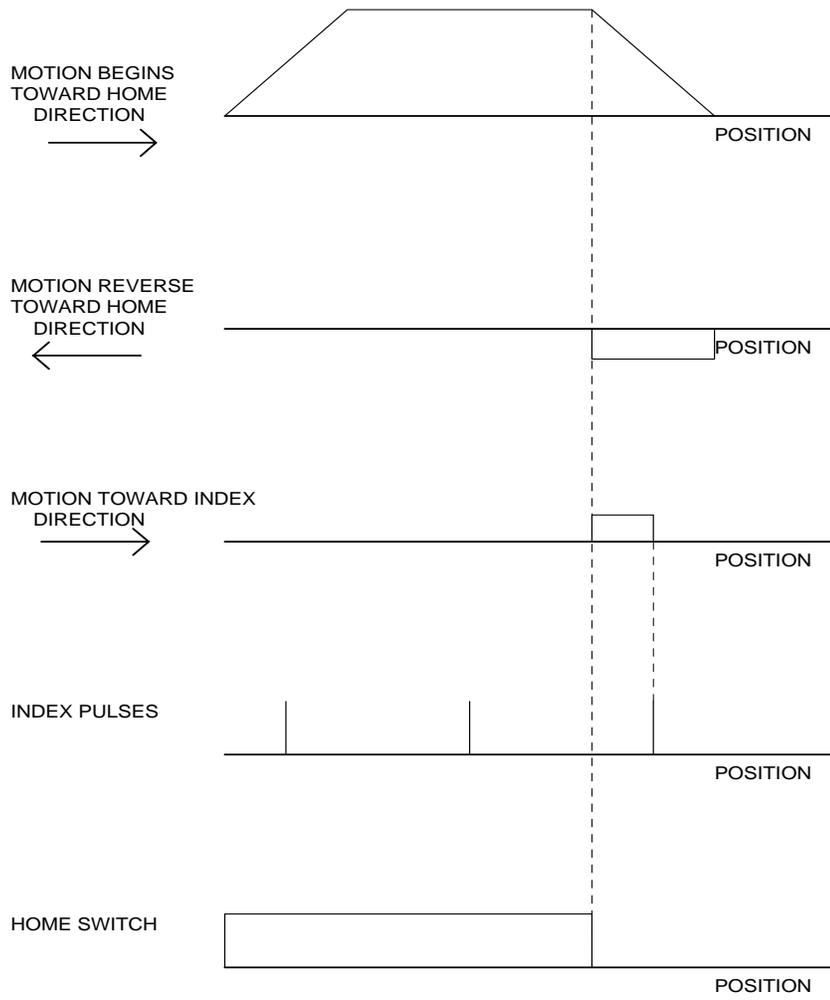
**Example - Jog in X only**

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.

<u>Instruction</u>	<u>Interpretation</u>
#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):**

<u>Instruction</u>	<u>Interpretation</u>
#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 #IMSWI special label subroutine will reverse the CN 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

<b><u>Instruction</u></b>	<b><u>Interpretation</u></b>
#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	
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**

<b>Instruction</b>	<b>Interpretation</b>
#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 occurred"	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.

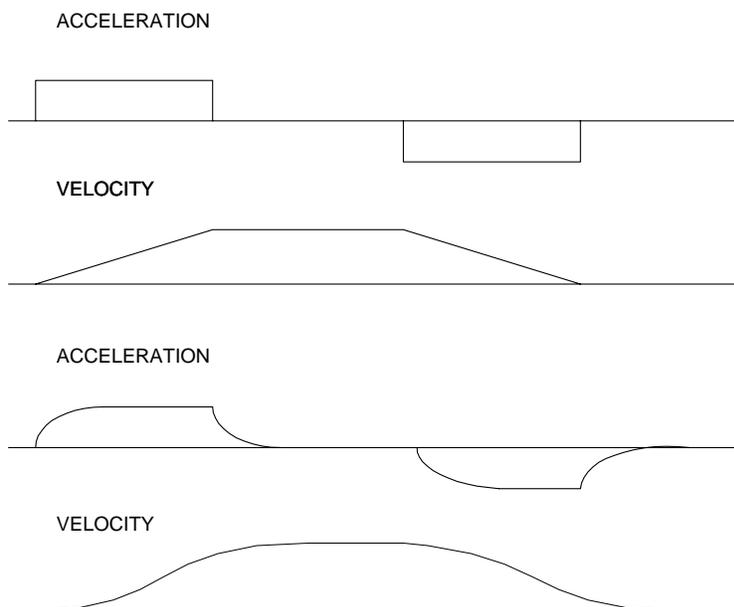
Instruction	Interpretation
#CONT	Label
AC 80000;DC 80000	Acceleration rate
JG 0	Start job mode
BGX	Start motion
#LOOP	
COMPP=@AN[1]*1000	Compute desired position
VE=COMPP-_TPX	Find position error
PVEL=VE*20	Compute velocity
JG PVEL	Change velocity
JP #LOOP	Repeat
EN	End

**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**

<u>Instruction</u>	<u>Interpretation</u>
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:***

<b><u>Instruction</u></b>	<b><u>Interpretation</u></b>
#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 LEGEND-MC**

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 a LEGEND-MC 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.

<b>Instruction</b>	<b>Interpretation</b>
#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	
#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 - Linear Move**

Make a coordinated linear move in the XY plane. Move to coordinates 40000,30000 counts at a vector speed of 100000 counts/sec and vector acceleration of 1000000 counts/sec<sup>2</sup>.

<u>Instruction</u>	<u>Interpretation</u>
LM XY	Specify axes for linear interpolation
LI40000,30000	Specify XY distances
LE	Specify end move
VS 100000	Specify vector speed
VA 1000000	Specify vector acceleration
VD 1000000	Specify vector deceleration
BGS	Begin sequence

**Example - Multiple Interpolated Moves**

This example makes a coordinated linear move in the XY plane. The Arrays VX and VY are used to store 750 incremental distances which are filled by the program #LOAD.

<b><u>Instruction</u></b>	<b><u>Interpretation</u></b>
#LOAD	Load Program
DM VX [750],VY [750]	Define Array
COUNT=0	Initialize Counter
N=0	Initialize position increment
#LOOP	LOOP
VX [COUNT]=N	Fill Array VX
VY [COUNT]=N	Fill Array VY
N=N+10	Increment position
COUNT=COUNT+1	Increment counter
JP #LOOP,COUNT<750	Loop if array not full
#A	Label
CAS	Specify coordinate system S
LM XY	Specify linear mode for XY
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 VX[COUNT],VY[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

**Example of Linear Interpolation Motion:**

<b><u>Instruction</u></b>	<b><u>Interpretation</u></b>
#LMOVE	Label
DP 0,0	Define position of X and Y 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

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.

### **Generating an Array - An Example**

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:

$$\omega = (A/B) [1 - \cos (2\pi T/B)]$$

$$X = (AT/B) - (A/2\pi)\sin (2\pi T/B)$$

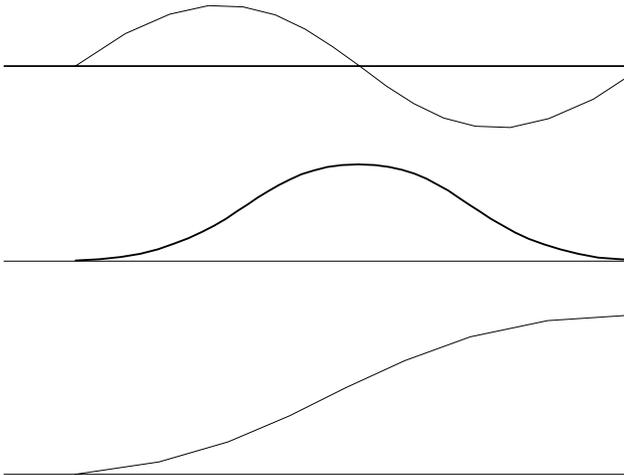
**Note:**  $\omega$  is the angular velocity;  $X$  is the position; and  $T$  is the variable, time, in milliseconds.

In the given example, A=6000 and B=120, the position and velocity profiles are:

$$X = 50T - (6000/2\pi) \sin (2\pi T/120)$$

Note that the velocity,  $\omega$ , 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 LEGEND-MC 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 LEGEND-MC)
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:***

<b><u>Instruction</u></b>	<b><u>Interpretation</u></b>
#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

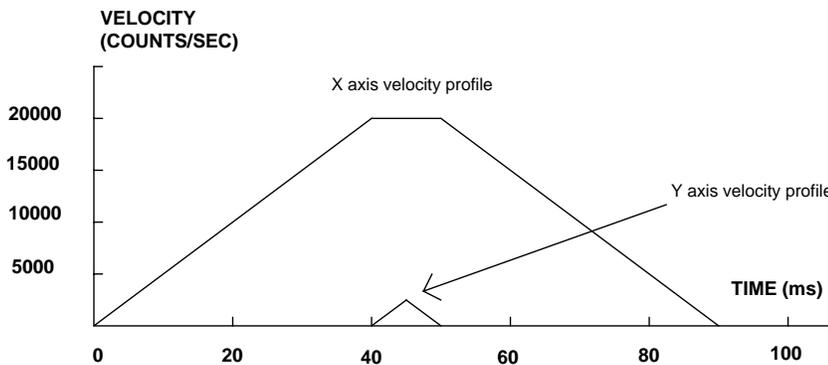
Required Motion Profiles:

X-Axis	2000 counts	Position	Y-Axis	100 counts	Position
	15000 count/sec	Speed		5000 count/sec	Speed
	500000 counts/sec <sup>2</sup>	Acceleration		500000 counts/sec <sup>2</sup>	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.

<u>Instruction</u>	<u>Interpretation</u>
#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 <sup>2</sup> for all axes
DC 500000,500000	Specify deceleration of 500000 counts / sec <sup>2</sup> 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 LEGEND-MC. High on input 1 means switch is ON.

<b>Instruction</b>	<b>Interpretation</b>
#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 a LEGEND-MC application program. #AUTO is usually the first line of a program. When this program is burned into the LEGEND-MC 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.

<b><u>Instruction</u></b>
#AUTO
ERX=150; OEX=1, I13
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.

<b><u>Instruction</u></b>
#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 AB1 command, the motors will continue doing whatever they were doing before the fault. For example, if they were jogging, they will continue jogging.

<b>Instruction</b>
#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 are 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 works 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.)

<b>Instruction</b>
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  $2\pi$  inches, and it corresponds to 4000 quadrature, one inch of travel equals:  $4000/2\pi = 637$  count/inch

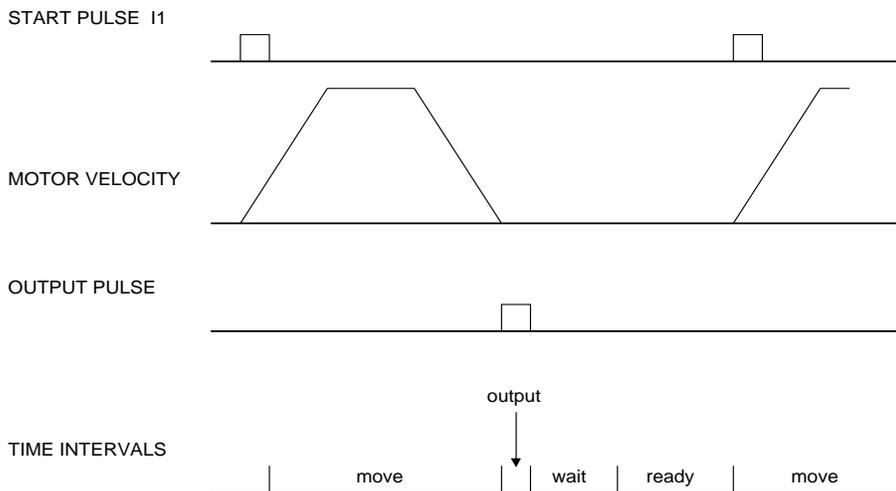
A distance of 10 inches equals 6370 counts, and a slew speed of 5 inches / second equals 3185 count/sec.

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 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:

$$3000 \text{ rpm} = 50 \text{ rev/sec} = 200000 \text{ count/sec}$$

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

$$\text{Speed} = 20000 \times \text{VIN}$$

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
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:

$$\text{Correction} = \text{Load Position Error} - \text{Rotary Position Error}$$

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:**

<b>Instruction</b>	<b>Function</b>
#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. Communication
3. Stability and Compensation
4. 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.

## Communication

Symptom	Cause	Remedy
LEGEND-MC cannot communicate with the controller.	Plug and Play installation did not proceed properly	Check first that LEGEND-MC.INF was used to install the controller. Next check the controller registry to see if the controller was automatically added and an address selected.

## 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.



**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 Paulo-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