
E Enable Communication

Type	Communication Interface	Product	Rev
Syntax	<i_><!>E	6K	5.0
Units	i = unit number set by the ADDR command		
Range	i = 0 - 99; b = 0 (serial communication off) or 1 (serial communication on)		
Default	b = 1		
Response	0_E: *E1		
See Also	ADDR, BAUD, DRPCHK, ECHO, LOCK, PORT, XONOFF		

The E command allows you to enable and disable serial ports on your 6K controller. To enable all units in the RS-232 daisy-chain or RS-485 multi-drop at one time, you can use the E1 command. The PORT command determines which COM port is affected by the E command.

Example:

```
PORT1          ; Next command affects the COM1 serial port on the 6K product
0_E1           ; Enable serial port for unit with device address 0
```

ECHO Communication Echo Enable

Type	Communication Interface	Product	Rev
Syntax	<!>ECHO	6K	5.0
Units	n/a		
Range	b = 0 (disable), 1 (enable), 2 (echo through other COM port), 3 (echo through both COM ports), or X (don't change)		
Default	1		
Response	ECHO: *ECHO0		
See Also], [, BAUD, EOL, EOT, ERRLVL, PORT, [SS], TSS		

The Communication Echo Enable (ECHO) command enables command echo. *Lower-case letters are converted to upper case and then echoed.* When echo is enabled, commands are echoed character by character.

In a terminal emulator mode, you may not see the echoed characters on your display when issuing commands that have a response, because the echoed characters may be overwritten by the response.

The PORT command determines which COM port is affected by the ECHO command.

The purpose of the ECHO2 and ECHO3 options is to accommodate an RS-485 multi-drop configuration in which a host computer communicates to the “master” 6K controller over RS-232 (COM1 port) and the master 6K controller communicates over RS-485 (COM2 port) to the rest of the units on the multi-drop. For this configuration, the echo setup should be configured by sending to the master the following commands executed in the order shown. In this example, it is assumed that the master's device address is set to 1. Hence, each command is prefixed with “1_” to address only the master unit.

```
1_PORT2       Subsequent command affects COM2, the RS-485 port
1_ECHO2       Echo characters back through the other port, COM1
1_PORT1       Subsequent command affects COM1, the RS-232 port
1_ECHO3       Echo characters back through both ports, COM1 and COM2
```

EFAIL Encoder Failure Detect

Type	Encoder Configuration	Product	Rev
Syntax	<!><@>EFAIL	6K	5.0
Units	n/a		
Range	b = 0 (disable), 1 (enable), or X (don't change)		
Default	0		
Response	EFAIL: *EFAIL0000_0000 1EFAIL: *1EFAIL0		
See Also	[ASX], [ER], ERROR, ERRORP, TASX, TER		

The Encoder Failure Detect (EFAIL) command enables (1) or disables (0) the monitoring of the encoder signals to determine if the encoder is functioning properly. If there is no active signal on either Phase A or Phase B of an axis encoder (i.e., encoder is fully disconnected), this will be detected and can elicit an appropriate response by programming the unit to monitor and recognize the encoder failure.

If EFAIL is enabled for an axis, and an encoder error is detected, then bit 5 of the extended axis status register (reported with TASX, TASXF and ASX) is set to 1. When ERROR bit 17 is set to 1, an encoder failure occurring on any axis will initiate a jump to the error program (ERRORP).

ELSE Else Condition of IF Statement

Type	Program Flow Control	Product	Rev
Syntax	<!>ELSE	6K	5.0
Units	n/a		
Range	n/a		
Default	n/a		
Response	n/a		
See Also	IF, NIF		

This command is used in conjunction with the IF and NIF commands to provide conditional branching. If the expression contained within the parentheses of the IF command evaluates true, then the commands between the IF and the ELSE are executed. The commands after the ELSE until the NIF are ignored. If the expression evaluates false, the commands between the ELSE and the NIF are executed. The commands between IF and ELSE are ignored. The ELSE command is **optional** and does not have to be included in the IF statement. IF()...ELSE...NIF commands can be nested up to 16 levels deep.

Programming order: IF(expression) ...commands... ELSE ...commands... NIF

Example:

```
IF(IN.1=b1)      ; Specify condition: if onboard input #1 is on
T5              ; If condition evaluates true, wait 5 seconds
ELSE            ; Else part of IF condition
TPE            ; If condition does not evaluate true transfer position of
              ; all encoders
NIF            ; End IF statement
```

ENCCNT Encoder Count Reference Enable

Type	Encoder; Controller Configuration	Product	Rev
Syntax	<!><@><a>ENCCNT	6K	5.0
Units	b = enable bit		
Range	0 (reference the commanded position), 1 (reference the encoder position) or X (don't change)		(applicable only to stepper axes)
Default	0 (reference the commanded position)		
Response	ENCCNT *ENCCNT0000_0000 1ENCCNT *1ENCCNT0		
See Also	AXSDEF, INFNC, LIMFNC, OUTP, [PCC], [PCE], [PCME], [PE], TPCC, TPCE, TPCME, TPE, TVELA, [VELA]		

Use ENCCNT to configure stepper axes to reference either the encoder position or the commanded position when capturing the position (see INFNCi-H) and checking the encoder position (PE and TPE). When checking the actual velocity (VELA and TVELA), ENCCNT determines whether the velocity, in units of revs/sec, is derived with the encoder resolution (ERES) or the drive resolution (DRES). The default setting (ENCCNT0) references the commanded position.

Example:

```
AXSDEF00      ; Axes 1 & 2 as steppers; axis 1 has encoder, but axis 2 does not.
INFNC1-H      ; Configure trigger 1A as a position capture input for axis 1
INFNC3-H      ; Configure trigger 2A as a position capture input for axis 2
ENCCNT10     ; Capture axis 1's encoder position when trigger 1A is activated,
              ; Capture axis 2's commanded position when trigger 2A is activated.
```

ENCPOL Encoder Polarity

Type	Encoder; Controller Configuration	Product	Rev
Syntax	<!><@><a>ENCPOL	6K	5.0
Units	b = polarity bit		
Range	0 (normal polarity), 1 (reverse polarity) or X (don't change)		
Default	0		
Response	ENCPOL *ENCPOL00000000 1ENCPOL *1ENCPOL0		
See Also	CMDDIR, EFAIL, [FB], FOLMAS, MEPOL, [PCE], [PE], [PER], PSET, SFB, TFB, TPE, TPCE, TPER		

Servo stability requires a direct correlation between the commanded direction and the direction of the encoder counts (i.e., a positive commanded direction from the controller must result in positive counts from the encoder).

If the encoder input is counting in the wrong direction, you may reverse the polarity with the ENCPOL command (see programming example below). This allows you to reverse the counting direction without having to change the actual wiring to the encoder input. For example, if the encoder on axis 2 counted in the wrong direction, you could issue the ENCPOLx1 command to correct the polarity.

Immediately after issuing the ENCPOL command, the encoder will start counting in the opposite direction (including all encoder position registers). For servo axes, the polarity is immediately changed whether or not encoder feedback is currently selected with the SFB command.

NOTE

Changing the feedback polarity effectively invalidates any existing offset position (PSET) setting; therefore, you will have to re-establish the PSET position.

The ENCPOL command is automatically saved in non-volatile RAM.

NOTE: ENCPOL does not affect the Master Encoder (the encoder connected to the “Master Encoder” connector). To change the polarity of the Master Encoder, use the MEPOL command.

If you wish to reverse the commanded direction of motion, first make sure there is a direct correlation between commanded direction and encoder direction, then issue the appropriate CMDDIR command to reverse both the commanded direction and the encoder direction (see CMDDIR command description for full details).

Example (servo axis):

```

SFB1      ; Select encoder feedback for axis 1
SMPER100  ; Set maximum position error to 100 units on axis 1
PSET0     ; Define current position of axis 1 as position zero
1TPE      ; *1TPE+0 (response indicates encoder #1 is at position zero)
MA0       ; Select incremental positioning mode
D+8000    ; Set distance to 8,000 units in the positive direction
GO1       ; Move axis 1. If the encoder polarity is incorrect, the axis will be
          ; unstable and will stop (drive disabled) as soon as the maximum
          ; position error of 100 units is reached.

1TPE      ; *1TPE-100 (response should show that encoder #1 is approximately at
          ; position -100; the minus sign indicates that the encoder is
          ; counting in the wrong direction)

ENCPOL1   ; Reverse encoder polarity on axis 1
PSET0     ; Define current position of axis 1 as position zero
DRIVE1    ; Enable the drive (drive was disabled when the SMPER value was
          ; exceeded)
D+8000    ; Set distance to 8,000 units in the positive direction
GO1       ; Move axis 1
1TPE      ; *1TPE+8000 (response shows encoder #1 has moved 8,000 units in the
          ; positive direction, indicating that the encoder is now counting in
          ; the correct direction)

```

ENCSDND Encoder Step and Direction Mode

Type	Encoder; Counter	Product	Rev
Syntax	<!><@><a>ENCSDND	6K	5.0
Units	n/a		
Range	b = 0 (quadrature signal), 1 (step & direction) or X (don't care)		
Default	0		
Response	ENCSDND: *ENCSDND0000_0000 1ENCSDND: *1ENCSDND0		
See Also	MESND, [PE], SFB, TPE		

Use the **ENCSDND** command to change the functionality of one or more of the encoder connectors to accept a counting source from a step and direction signal, instead of from an encoder quadrature signal. The counter is reported by **PE** and **TPE**. If the axis is a servo axis, the step and direction count source is used even though the feedback source selected (**SFB**) is an “encoder.”

ENCSDND0.....(default setting) accept a quadrature signal from an encoder.

ENCSDND1.....Accept step and direction signals. The count is registered on a positive edge of a transition for a signal measured on encoder channel A+ and A- connections. The direction of the count is specified by the signal on encoder channel B+ and B- connections. Therefore, you should connect your step and direction input device as follows: Connect Step+ to A+, Step- to A-, Direction+ to B+, and Direction- to B-.

END End Program/Subroutine/Path Definition

Type	Program or Subroutine Definition	Product	Rev
Syntax	<!>END	6K	5.0
Units	n/a		
Range	n/a		
Default	n/a		
Response	n/a		
See Also	\$, DEF, DEL, ERASE, GOBUF, GOSUB, GOTO, RUN		

The **END** command marks the ending point of a program/subroutine/path contour definition. All commands between the **DEF** and the **END** statement will be considered in a program, subroutine, or path contour.

Example:

```

DEF pick      ; Begin definition of program named pick
GO1100       ; Initiate motion on axes 1 and 2
END          ; End program definition
pick         ; Execute program named pick

```

EOL End of Line Terminating Characters

Type	Communication Interface	Product	Rev
Syntax	<!>EOL<i>,<i>,<i>	6K	5.0
Units	n/a		
Range	i = 0 - 256		
Default	13,10,0		
Response	EOL: *EOL13,10,0		
See Also], [, BOT, EOT, ERRLVL, PORT, WRITE, XONOFF		

The End of Line Terminating Characters (EOL) command designates the characters to be placed at the end of each line, but not the last line, in a multi-line response. The last line of a multi-line response has the EOT characters. Up to 3 characters can be placed at the end of each line. The characters are designated with their ASCII equivalent (no character that has a value of zero [Ø] will be output). For example, a carriage return is ASCII 13, a line feed is ASCII 10, and no terminating character is designated with a zero.

The PORT command determines which COM port is affected by the EOL command.

NOTE: Although you may issue a single command, like TSTAT, each line of the response will have the EOL characters. The last line in the response will have the EOT characters. If the response is only one line long, the EOT characters will be placed after the response, not the EOL characters.

Character	ASCII Equivalent
Line Feed	10
Carriage Return	13
Ctrl-Z	26

For a more complete list of ASCII Equivalents, refer to the ASCII Table in Appendix B.

Example:
EOL13,0,0 ; Place a carriage return after each line of a response

EOT End of Transmission Characters

Type	Communication Interface	Product	Rev
Syntax	<!>EOT<i>,<i>,<i>	6K	5.0
Units	n/a		
Range	i = 0 - 256		
Default	13,0,0		
Response	EOT: *EOT13,0,0		
See Also], [, BOT, EOL, ERRLVL, PORT, WRITE		

The End of Transmission Terminating Characters (EOT) command designates the characters to be placed at the end of every response. Up to 3 characters can be placed after the last line of a multi-line response, or after all single-line responses. The characters are designated with their ASCII equivalent (no character that has a value of zero [Ø] will be output). For example, a carriage return is ASCII 13, a line feed is ASCII 10, a Ctrl-Z is ASCII 26, and no terminating character is designated with a zero.

The PORT command determines which COM port is affected by the EOT command.

NOTE: Although you may issue a single command, like TSTAT, each line of the response will have the EOL characters. The last line in the response will have the EOT characters. If the response is only one line long, the EOT characters will be placed after the response, not the EOL characters.

Character	ASCII Equivalent
Line Feed	10
Carriage Return	13
Ctrl-Z	26

For a more complete list of ASCII Equivalents, refer to the ASCII Table in Appendix B.

Example:
EOT13,10,26 ; Place a carriage return, line feed, and Ctrl-Z after the last line
; of a multi-line response, and after all single line responses

[ER]

Error Status

Type	Assignment or Comparison	Product	Rev
Syntax	See below	6K	5.0
Units	n/a		
Range	n/a		
Default	n/a		
Response	n/a		
See Also	[ASX], DRFEN, DRFLVL, EFAIL, ERROR, ERRORP, ESTALL, GOWHEN, INFNC, K, LH, LIMFNC, LS, S, SMPER, STRGTT, TASX, TER, TERF		

The Error Status (ER) command is used to assign the error status bits to a binary variable, or to make a comparison against a binary or hexadecimal value. To make a comparison against a binary value, the letter b (b or B) must be placed in front of the value. The binary value itself must only contain ones, zeros, or Xs (1, 0, X, x). To make a comparison against a hexadecimal value, the letter h (h or H) must be placed in front of the value. The hexadecimal value itself must only contain the letters A through F, or the numbers 0 through 9.

Syntax: VARBn=<i%>ER where n is the binary variable number, or ER can be used in an expression such as IF(ER=b1101), or IF(ER=h7F). **NOTE:** If you are using multi-tasking, be aware that each task has its own error status register. If you wish to check the error status of an external task (a task other than the task that is executing the ER operator), then you must prefix the ER operator to address the targeted task (e.g., 2%ER for the error status of Task 2).

The bit select operator (.), in conjunction with the bit number, can be used to specify a specific error bit. Examples: VARB1=ER.2 assigns error bit 2 to binary variable 1; IF(ER.2=b1) is a conditional statement that is true if error bit 2 is set to 1.

The specific error-checking bits must be enabled by the Error-Checking Enable (ERROR) command before the ER command will provide an error response — see programming example below.

Multi-Tasking: If you are using multi-tasking, each task has its own error checking bits (ERROR), error status register (ER, TER, TERF), and ERRORP program. Regarding axis-related error conditions (e.g., drive fault, end-of-travel limit, etc.), only errors on the task's associated (TSKAX) axes are detected in its error status register.

The function of each error status bit is shown below.

Bit #	Function (1 = Yes; 0 = No)
1*	Stall Detected: Functions when stall detection has been enabled (ESTALL).
2	Hard Limit Hit: Functions when hard limits are enabled (LH).
3	Soft Limit Hit: Functions when soft limits are enabled (LS).
4	Drive Fault: Detected only if the drive is enabled (DRIVE), the drive fault input is enabled (DRFEN), and the drive fault level is set correctly (DRFLVL).
5	RESERVED (refer to the ERROR command)
6	Kill Input: When an input is defined as a Kill input (INFNCi-C or LIMFNCi-C), and that input becomes active.
7	User Fault Input: When an input is defined as a User Fault input (INFNCi-F or LIMFNCi-F), and that input becomes active.
8	Stop Input: When an input is defined as a Stop input (INFNCi-D or LIMFNCi-D), and that input becomes active.
9	Enable input is activated (not grounded).
10	Pre-emptive (on-the-fly) GO or registration move profile not possible.
11**	Target Zone Settling Timeout Period (set with the STRGTT command) is exceeded.
12**	Maximum Position Error (set with the SMPER command) is exceeded.
13	RESERVED
14	Position relationship in GOWHEN already true when GO, GOL, FSHFC, or FSHFD was executed.
15	RESERVED
16	Bad command detected (bit is cleared with TCMDER command).
17	Encoder failure (EFAIL1 must be enabled before error can be detected; error is cleared by sending EFAIL0 to the affected axis).
18	Cable to an expansion I/O brick is disconnected, or power to the I/O brick is lost; error is cleared by reconnecting the I/O brick and issuing the ERROR.18-0 command and then the ERROR.18-1 command.
19-32	RESERVED

* Stepper axes only

** Servo axes only

Example:

```

ERROR111101101 ; Enable error-checking bits 1-4, 6, 7, and 9
VARB1=ER        ; Error status assigned to binary variable 1
VARB2=ER.4      ; Error status bit 4 assigned to binary variable 2
VARB2           ; Response if bit 4 is set to 1:
                ; *VARB2=XXX1_XXXX_XXXX_XXXX_XXXX_XXXX_XXXX_XXXX
IF(ER=b1110X11X1) ; If the error status contains 1's for bit locations 1, 2, 3,
                ; 6, 7, and 9, and a 0 for bit location 4, do the IF statement
TREV           ; Transfer revision level
NIF           ; End if statement
IF(ER=hF600)   ; If the error status contains 1's for bit locations 1, 2, 3,
                ; 4, 6, and 7, and 0's for every other bit location, do the
                ; IF statement
TREV           ; Transfer revision level
NIF           ; End if statement

```

ERASE Erase All Programs

Type	Subroutine or Program Definition	Product	Rev
Syntax	<!>ERASE	6K	5.0
Units	n/a		
Range	n/a		
Default	n/a		
Response	n/a		
See Also	[DATP], DEF, DEL, RESET		

The Erase All Programs (ERASE) command deletes all programs created with the DEF command, including all data programs (DATP). If you do not want to erase all the programs, you can use the DEL command to selectively delete programs. The RESET command will erase all programs (only in bus-based controllers) and reset all values to factory defaults.

ERES Encoder Resolution

Type	Encoder Configuration	Product	Rev
Syntax	<!><@><a>ERES<i>,<i>,<i>,<i>,<i>,<i>,<i>,<i>,<i>	6K	5.0
Units	i = counts/rev		
Range	1 - 65535 (stepper axes); 200 - 1024000 (servo axes)		
Default	4000		
Response	ERES: *ERES4000,4000,4000,4000,4000,4000,4000,4000 1ERES: *1ERES4000		
See Also	DRES, EFAIL, ENCCNT, ESTALL, TSTAT		

Stepper Axes: The ERES command establishes the number of encoder counts received for a move equal to the distance set in the drive resolution (DRES) command. If the motor/drive resolution equals 25000 steps/rev, and a 1 revolution move is performed (with scaling (SCALE) disabled), the number of encoder counts received back would be the encoder resolution value (ERES). A standard 1000-line per revolution encoder gives 4000 counts post-quadrature. If the encoder is coupled to the back of a motor, the ERES value will be 4000. This value, along with the drive resolution value (DRES) are important for the motion algorithm to correctly interpret move distances, move velocities, and move accelerations.

Servo Axes: The servo system's resolution is determined by the resolution of the encoder used with the servo drive/motor system. The ERES command establishes the number of steps, or *counts* (post quadrature), per unit of travel. For example, Compumotor's SM and NeoMetric Series motors with the "E" encoder option use 1,000-line encoders, and therefore have a 4,000 count/rev post-quadrature resolution (requires ERES4000). If the encoder is mounted directly to the motor, then to ensure that the motor will move according to the programmed distance and velocity, the controller's resolution (ERES value) must match the encoder's resolution.

Resolutions for Compumotor Encoders:

Stepper axes:

- -RE, -RC, -EC, and -E Series Encoders: ... ERES4000
- -HJ Series Encoders: ERES2048

Servo axes:

- SM, N or J Series Servo Motors:..... SM/N/JxxxxD-xxxx: ERES2000
SM/N/JxxxxE-xxxx: ERES4000

Dynaserv (stepper or servo):

- DR10xxB..... ERES507904
- DR1xxxE..... ERES614400
- DR1xxxA..... ERES819200
- DR5xxxB..... ERES278528
- DR5xxxA..... ERES425894
- DM10xxB..... ERES655360
- DM1xxxA..... ERES1024000
- DM1004x ERES655360

Example (axis 1 is stepper axis, axis 2 is servo axis):

```
SCALE0          ; Disable scaling
DEL proga       ; Delete program called proga
DEF proga       ; Begin definition of program called proga
DRES25000       ; Motor/drive resolution set to 25000 steps/rev on axis 1
                ; (DRES is used for stepper axes only)
ERES4000,4000   ; Encoder resolution set to 4000 post-quadrature counts/rev on
                ; axes 1 & 2 (encoder on axis 1 is for stall detection only)
ESTALL1         ; Enable stall detection for the stepper axis (axis 1)
MA00            ; Incremental mode for axes 1 and 2
MC00            ; Preset mode for axes 1 and 2
A10,12         ; Set the acceleration to 10 and 12 units/sec/sec for axes 1 & 2
V1,1           ; Set the velocity to 1 unit/sec for axes 1 and 2
D100000,80000  ; Set the distance to 100000 and 1000 units for axes 1 and 2
GO11           ; Initiate motion on axes 1 and 2:
                ; Axis 1 will move 100,000 commanded counts (4 revs).
                ; Axis 2 will move 80,000 encoder counts (20 revs)
END            ; End definition of proga
```

ERRBAD Error Prompt

Type	Communication Interface	Product	Rev
Syntax	<!><@>ERRBAD<i>,<i>,<i>,<i>	6K	5.0
Units	n/a		
Range	i = 0 - 256		
Default	13,10,63,32		
Response	ERRBAD: *ERRBAD13,10,63,32		
See Also	BOT, EOT, ERDEF, ERRLVL, ERROK, PORT, TCMER		

The Error Prompt (ERRBAD) command designates the characters to be placed into the output buffer after an erroneous command has been entered. Up to 4 characters can be placed in the output buffer. These characters serve as a prompt for the next command. The characters are designated with their ASCII equivalent. For example, a carriage return is ASCII 13, a line feed is ASCII 10, a question mark is ASCII 63, a space is ASCII 32, and no terminating character is designated with a zero.

The PORT command determines which COM port is affected by the ERRBAD command.

For a more complete list of ASCII equivalents, refer to the ASCII Table in Appendix B.

Example:

```
ERRBAD13,0,0,0 ; Place a carriage return only in the output buffer after
                ; processing an erroneous command
```

ERRDEF Program Definition Prompt

Type	Communication Interface	Product	Rev
Syntax	<!><@>ERRDEF<i>,<i>,<i>,<i>	6K	5.0
Units	n/a		
Range	i = 0 - 256		
Default	13,10,45,32		
Response	ERRDEF: *ERRDEF13,10,45,32		
See Also	ERRBAD, ERRLVL, ERROK, PORT, XONOFF		

The Program Definition Prompt (ERRDEF) command designates the characters to be placed into the output buffer after a DEF command has been entered. These characters will continue to be placed into the output buffer after each command until the END command is processed. Up to 4 characters can be placed in the output buffer. These characters serve as a prompt while defining a program. The characters are designated with their ASCII equivalent. For example, a carriage return is ASCII 13, a line feed is ASCII 10, a hyphen is ASCII 45, a space is ASCII 32, and no terminating character is designated with a zero. For a more complete list of ASCII equivalents, refer to the ASCII Table in Appendix B.

The PORT command determines which COM port is affected by the ERRDEF command.

Example:

```
ERRDEF13,0,0,0 ; Place a carriage return only in the output buffer after each  
                ; command in the program definition
```

ERRLVL Error Detection Level

Type	Error Handling	Product	Rev
Syntax	<!>ERRLVL<i>	6K	5.0
Units	i - error level settings		
Range	i = 0, 1, 2, 3, or 4		
Default	4 if COM port is set up for RS-232C; 0 if COM port is set up for RS-485		
Response	ERRLVL: *ERRLVL4		
See Also	EOT, ERRBAD, ERRDEF, ERROK, PORT		

The Error Detection Level (ERRLVL) command specifies the format for all **Response** feedback and error messages (error messages are listed on page 9 of this manual, and in the Troubleshooting chapter of the *Programmer's Guide*). Error level 4 is the default error detection level.

The PORT command determines which COM port is affected by the ERRLVL command.

Error Level	Description
ERRLVL4	All responses are returned as shown in the Response field of the corresponding command, followed by the EOT characters and the ERROK characters. Error conditions return an error message corresponding to the error condition followed by the EOT characters and the ERRBAD characters. Program definitions beginning with the DEF command and ending with the END command place the ERRDEF characters in the output buffer after each command is processed.
ERRLVL3	All responses are returned as shown in the Response field of the corresponding command, followed by the EOT characters and the ERROK characters. Error conditions return only the ERRBAD characters. Program definitions beginning with the DEF command and ending with the END command place the ERRDEF characters in the output buffer after each command is processed.
ERRLVL2	All responses are returned as shown in the Response field of the corresponding command, followed by the EOT characters. There are no ERROK characters and no error responses.
ERRLVL1	All responses are returned as shown in the Response field of the corresponding command, minus the command itself, followed by the EOT characters. There is no error response.
ERRLVL0	All responses are returned as shown in the Response field of the corresponding command, minus the command itself and the asterisk, followed by the EOT characters. There is no error response.

Bit #	Function (Error bits #13, #15, and #18 - #32 are reserved.)	Branch Type
6	Input Kill: When an input is defined as a KILL input (INFNCi-C or LIMFNCi-C), and that input becomes active.	GOTO
7	User Fault Input: When an input is defined as a user fault input (INFNCi-F or LIMFNCi-F), and that input becomes active.	GOTO
8	Stop Input: When an input is defined as a stop input (INFNCi-D or LIMFNCi-D), and that input becomes active.	GOTO
9	Enable input is activated (not grounded).	GOTO
10	Pre-emptive (on-the-fly) GO or registration move profile not possible at the time of attempted execution.	GOSUB
11 **	Target Zone Settling Timeout Period (set with the STRGTT command) is exceeded.	GOSUB
12 **	Maximum Position Error (set with the SMPER command) is exceeded.	GOSUB
14	GOWHEN condition already true when a subsequent GO, GOL, FSHFC, or FSHFD command is executed.	GOSUB
16	Bad command detected (bit is cleared with TCMER command).	GOSUB
17	Encoder failure (EFAIL1 must be enabled before error can be detected; error is cleared by sending EFAIL0 to the affected axis).	GOSUB
18	Cable to an expansion I/O brick is disconnected, or power to the I/O brick is lost; error is cleared by reconnecting the I/O brick (or restore power to the I/O brick) and issuing the ERROR.18-0 command and then the ERROR.18-1 command.	GOTO

* Stepper axes only; ** Servo axes only

NOTE: Error bits 13, 15, and 19-32 are reserved.

ERRORP Error Program Assignment

Type	Error Handling	Product	Rev
Syntax	<!><%>ERRORP<t>	6K	5.0
Units	t = text (name of error program)		
Range	Text name of 6 characters or less		
Default	n/a		
Response	ERRORP: *ERRORPerr1		
See Also	[ER], ERRLVL, ERROR, TER		

Using the ERRORP command, you can assign any previously defined program as the error program. For example, to assign a previously defined program named CRASH as the error program, enter the ERRORP CRASH command. If you later decide not to have an error program, issue the ERRORP CLR command; after the ERRORP CLR command, no error program will be called until you assign a new one.

The purpose of the error program is to provide a programmed response to certain error conditions (see table below) that may occur during the operation of your system. Programmed responses typically include actions such as shutting down the drive(s), activating or de-activating outputs, etc. To detect and respond to the error conditions, the corresponding error-checking bit(s) must be enabled with the ERROR command (refer to the *ERROR Bit #* column in the table below). It is the programmer's responsibility to determine the cause of the error, and take action based on the error. The error condition can be determined using the ER evaluation in an IF statement (e.g., IF (ER=b10X)). An error program set-up example is provided in the *Programmer's Guide*.

When an error condition occurs and the associated error-checking bit has been enabled with the ERROR command, the 6K controller will branch to the error program. Depending on the error condition, the branch be either a GOTO or GOSUB. If the error condition calls for a GOSUB, then after the ERRORP program is executed, program control returns to the point at which the error occurred. If you do not want to return to the point at which the error occurred, you can use the HALT command to end program execution or you can use the GOTO command to go to a different program. If the error condition calls for a GOTO, there is no way to return to the point at which the error occurred.

MULTI-TASKING

If you are operating multiple tasks, be aware that you must enable error conditions (`ERROR`) and specify an error program (`ERRORP`) for each task (e.g., `2%ERROR.2-1` and `2%ERRORP.FIX` for Task 2). Each task has its own error status register (reported with `ER`, `TER`, and `TERF`). Regarding axis-related error conditions (e.g., drive fault, end-of-travel limit, etc.), only errors on the task's associated (`TSKAX`) axes will cause a branch to the task's `ERRORP` program.

The `ERRORP` assignment is not saved in battery-backed RAM. To ensure that the `ERRORP` assignment is retained when you cycle power or issue a `RESET` command, put the `ERRORP` command in the *startup* program assigned with the `STARTP` command.

WHEN TO BRANCH

If you wish the branch to the error program to occur at the time the error condition is detected, use the continuous command execution mode (`COMEXCL`). Otherwise, the branch will not occur until motion on all axes has stopped.

Canceling the Branch to the Error Program: The error program will be continuously called/repeated until you cancel the branch to the error program. (This is true for all cases except error condition #9, enable input activated, in which case the error program is called only once.) There are three ways to cancel the branch:

- Disable the error-checking bit with the `ERROR.n-0` command, where "n" is the number of the error-checking bit you wish to disable. For example, to disable error checking for the kill input activation (bit #6), issue the `ERROR.6-0` command. To re-enable the error-checking bit, issue the `ERROR.n-1` command.
- Delete the program assigned as the `ERRORP` program (`DEL <name of program>`).
- Satisfy the *How to Remedy the Error* requirement identified in the table below.

NOTE

In addition to canceling the branch to the error program, you must also remedy the cause of the error; otherwise, the error program will be called again when you resume operation. Refer to the *How to Remedy the Error* column in the table below for details.

ERROR Bit #	Cause of the Error	Branch Type to <code>ERRORP</code>	How to Remedy the Error
1	Stepper axes only: Stall detected (Stall Detection and Kill On Stall must be enabled first—see <code>ESTALL</code> and <code>ESK</code> , respectively)	Gosub	Issue a <code>GO</code> command.
2	Hard Limit Hit (hard limits must be enabled first—see <code>LH</code>)	If <code>COMEXL0</code> , then Goto; If <code>COMEXL1</code> , then Gosub	Change direction & issue <code>GO</code> command on the axis that hit the limit; or issue <code>LH0</code> .
3	Soft Limit Hit (soft limits must be enabled first—see <code>LS</code>)	If <code>COMEXL0</code> , then Goto; If <code>COMEXL1</code> , then Gosub	Change direction & issue <code>GO</code> command on the axis that hit the limit; or issue <code>LS0</code> .
4	Drive Fault (Detected only if drive enabled – <code>DRIVE</code> , drive fault input enabled – <code>DRFEN</code> , and drive fault level correct – <code>DRFLVL</code>).	Goto	Clear the fault condition at the drive, & issue a <code>DRIVE1</code> command for the faulted axis.
5	Commanded Stop or Kill (whenever a <code>K</code> , <code>!K</code> , <code><ctrl>K</code> , <code>S</code> , or <code>!S</code> command is sent) — See “Commanded Kill or Stop” note below.	If <code>!K</code> , then Goto; If <code>!S & COMEXS0</code> , then Goto; If <code>!S & COMEXS1</code> , then Gosub, but need <code>!C</code>	No fault condition is present—there is no error to clear. <div style="border: 1px solid black; padding: 5px; width: fit-content; margin: 5px auto;">If you want the program to stop, you must issue the <code>!HALT</code> command.</div>
6	Kill Input Activated (see <code>INFNCi-C</code> or <code>LIMFNCi-C</code>)	Goto	Deactivate the kill input.

ERROR Bit #	Cause of the Error	Branch Type to ERRORP	How to Remedy the Error
7	User Fault Input Activated (see INFNCi-F or LIMFNCi-F)	Goto	Deactivate the user fault input, or disable it by assigning it a different function.
8	Stop Input Activated (see INFNCi-D or LIMFNCi-D)	Goto	Deactivate the stop input, or disable it by assigning it a different function.
9	Enable input not grounded	Goto	Re-ground the enable input, and issue a @DRIVE1 command.
10	Pre-emptive (on-the-fly) GO or registration move profile not possible at the time of attempted execution.	Gosub	Issue another GO command.
11	Servo Axes Only: Target Zone Timeout (STRGTT value has been exceeded).	Gosub	Issue these commands in this order: STRGTE0, D0, GO, STRGTE1
12	Servo Axes Only: Exceeded Max. Allowable Position Error (set with the SMPER command).	Gosub	Issue a DRIVE1 command to the axis that exceeded the allowable position error. Verify that feedback device is working properly.
14	GOWHEN condition already true when GO, GOL, FSHFC, or FSHFD executed.	Goto	Issue another GOWHEN command; or issue a !K command and check the program logic (use the TRACE and STEP features if necessary).
16	Bad command detected.	Gosub	Issue the TCMDEP command.
17	Encoder failure (EFAIL1 must be enabled before error can be detected).	Gosub	Send the EFAIL0 command to the affected axis.
18	Expansion I/O brick disconnected, or lost power.	Goto	Reconnect I/O brick or restore power. Then issue ERROR.18-0 and then ERROR.18-1.

Reserved Bits: Bits 13, 15, and 19-32 are reserved.

Branching Types: If the error condition calls for a GOSUB, then after the ERRORP program is executed, program control returns to the point at which the error occurred. If you do not want to return to the point at which the error occurred, you can use the HALT command to end program execution or you can use the GOTO command to go to a different program. If the error condition calls for a GOTO, there is no way to return to the point at which the error occurred.

Commanded Kill or Stop: When ERROR bit 5 is enabled (ERROR.5-1), a Stop (S or !S) or a Kill (K, !K or <ctrl>K) command will cause the controller to branch to the error program. Note, however, that this error condition does not set an error bit (ER), because there is no way to clear the error condition upon leaving the error program. Therefore, you should use the IF (ER=b00000000000000000000000000000000) statement in your error program to determine if the cause of the error was a commanded kill or stop (i.e., if no error bits are set).

Example:

```

DEF err1          ; Define error program err1
IF(ER=b01)       ; If the error is a hard limit, send a message & stop program
                  ; execution
WRITE"Hard Limit Hit" ; Write Hard Limit Hit message
HALT              ; Terminate program execution
NIF              ; End IF statement
IF(ER=b0X1)      ; If the error is a soft limit, back off the soft limit,
                  ; reset position, & continue
D~,~,~,~        ; Change direction in preparation to back off the soft limit
D1,1,1,1        ; Set distance to 1 step (just far enough to back off the soft
                  ; limit)
GO1111          ; Initiate the 1-step move to back off the soft limit
PSET0,0,0,0     ; Reset the position to zero
NIF              ; End IF statement
END              ; End definition of error program err1
ERRORP err1     ; Set error program to err1. Branch to err1 upon receiving a hard
                  ; or soft limit
ERROR01100000   ; Set error condition bits to look for hard limit or a soft limit

```

ESDB Stall Backlash Deadband

Type	Encoder Configuration	Product	Rev
Syntax	<!><@><a>ESDB<i>,<i>,<i>,<i>,<i>,<i>,<i>,<i>	6K	5.0
Units	i = encoder steps		
Range	0 - 99,999,999		(applicable only to stepper axes)
Default	0		
Response	ESDB: *ESDB0,0,0,0,0,0,0,0 1ESDB: *1ESDB0		
See Also	[AS], DRES, EFAIL, ERES, ESK, ESTALL, TAS		

The Stall Backlash Deadband (ESDB) command establishes the maximum number of encoder steps that a move can fall behind after a change in direction before stall detection is initiated. If there is no change in direction, the stall backlash deadband value will not be used to determine if there is a stall condition. To use the stall backlash deadband, stall detection (ESTALL) must be enabled.

A stall condition will be recorded by bit 12 of the axis status register. The TAS command can be used to get the axis status response.

Example: Refer to the enable stall detect (ESTALL) command example.

ESK Kill on Stall

Type	Encoder Configuration	Product	Rev
Syntax	<!><@><a>ESK	6K	5.0
Units	n/a		
Range	b = 0 (disable), 1 (enable), or X (don't change)		(applicable only to stepper axes)
Default	0		
Response	ESK: *ESK0000_0000 1ESK: *1ESK0		
See Also	DRES, EFAIL, ERES, ESDB, ESTALL		

The Kill on Stall (ESK) command will immediately stop pulses from being sent to an axis when a stall has been detected. Stall detect (ESTALL) must also be enabled before the ESK command will have any affect.

Example: Refer to the enable stall detect (ESTALL) command example.

ESTALL Enable Stall Detect

Type	Encoder Configuration	Product	Rev
Syntax	<!><@><a>ESTALL	6K	5.0
Units	n/a		
Range	b = 0 (disable), 1 (enable), or X (don't change)		(applicable only to stepper axes)
Default	0		
Response	ESTALL: *ESTALL0000_0000 1ESTALL: *1ESTALL0		
See Also	[AS], DRES, EFAIL, ENCCNT, [ER], ERES, ESDB, ESK, TAS, TER		

The Enable Stall Detect (ESTALL) command determines if stall conditions will be checked.

A stall condition will occur if the actual number of encoder counts received is less than expected for each motor step output segment. The number of encoder counts expected is determined by dividing the encoder resolution (ERES) by 100. The motor step output segment is determined by dividing the drive resolution (DRES) by 50.

For example, given an encoder resolution (ERES) of 4000 and a drive resolution (DRES) of 25000, the number of encoder counts expected for each motor step output segment = $\frac{4000}{100} = 40$. The motor step output segment = $\frac{25000}{50} = 500$. Therefore, during a move, after every 500 motor steps are sent out, the controller checks to see if it received 40 encoder counts. If it did, then everything is O.K. If not, then a stall condition exists.

When a stall condition occurs, it is reported in bit 12 in the AS and TAS axis status commands.

To accurately detect a stall, the drive resolution (DRES) and the encoder resolution (ERES) must be properly set.

Example:

```

SCALE0          ; Disable scaling
DEL progA       ; Delete program called progA
DEF progA       ; Begin definition of program called progA
DRES25000,25000 ; Motor/drive resolution set to 25000 steps/rev on axes 1 and 2
ERES4000,4000   ; Encoder resolution is 4000 post-quadrature counts/rev, both axes
ENCCNT11       ; Use encoder count references for axes 1 and 2
ESDB10,10      ; Stall backlash set to 10 commanded counts on axes 1 and 2
ESTALL11       ; Enable stall detection on axes 1 and 2
ESK11          ; Enable kill on stall for axes 1 and 2
MA00           ; Incremental positioning mode for axes 1 and 2
MC00           ; Preset positioning mode for axes 1 and 2
A10,12         ; Set the acceleration to 10 and 12 units/sec/sec for axes 1 and 2
V1,1           ; Set the velocity to 1 unit/sec for axes 1 and 2
D100000,250000 ; Set the distance to 100000 and 1000 units for axes 1 and 2
GO11           ; Initiate motion on axes 1 and 2:
                ; Axis 1 will move 100000 commanded counts (4 revs)
                ; Axis 2 will move 250000 commanded counts (10 revs)
                ; (If, at any time during the above moves any of the actual
                ; encoder counts fall behind, a stall condition will be flagged,
                ; and motion will stop on the appropriate axis.)
END            ; End definition of program progA

```

EXE**Execute a Program From a Compiled Program**

Type	PLC Scan Program	Product	Rev
Syntax	i%EXEt	6K	5.0
Units	i = Task Number t = Program Name (6 characters or less)		
Range	i = 1-10		
Default	n/a		
Response	n/a		
See Also	INSELP, PCOMP, PEXE, PLCP, SCANP		

Use the EXE command to start a standard (non-compiled) program from within a compiled PLCP program. The EXE command specifies the name of the program, and the task in which it will be launched. The program named in the EXE command need not be defined at the time the PLCP program is compiled; however, the program must be defined before the SCANP or PRUN is issued. If no task number is assigned with a % prefix, then the task in which the PLCP program is compiled (PCOMP) will be the task that runs the program. Note, however, that the EXE program cannot be executed in the Task Supervisor (task 0).

The PLCP program will ignore the EXE command if a currently running program is detected within the specified task; therefore, the EXE command can essentially only be used to initiate a new task with the program it is launching. Like the INSELP command, the program launched by the EXE command will not interrupt a currently running program, nor will it interrupt a WAIT or T command.

CAUTION: Using the SCANP command to run a PLCP program in Scan mode will cause the PLCP program to execute as often as every system update period (2 ms). An EXE command used within a PLCP program running in Scan mode could therefore attempt to launch a program in the specified task as often as every 2 ms. This may not allow enough time for the program launched in the specified task by the EXE command to complete before the same EXE command is issued again. As stated, the PLCP program will ignore the EXE command if a currently running program is detected, so timing must be considered when launching programs with the EXE command.

To execute a compiled program from within a compiled PLCP program, use the PEXE command.

Example:

```

DEF PLCP1      ; Define PLC program PLCP1
IF(IN.1=b1)    ; If input 1 is active
3%EXE PROG1    ; Launch program PROG1 in Task 3
ELSE
2%EXE PROG2    ; Otherwise launch program PROG2 in Task 2
NIF
END

PCOMP PLCP1    ; Compile PLCP1
SCANP PLCP1    ; Scan with program PLCP1

```

[FB]		Value of Current Feedback Device	
Type	Servo; Assignment or Comparison	Product	Rev
Syntax	See below	6K	5.0
Units	See below		
Range	See below		
Default	n/a		
Response	n/a		
See Also	[ANI], ANIFB, ANIRNG, CMDDIR, ENCPOL, GOWHEN, [PANI], [PE], PSET, SCALE, SCLD, SFB, TFB, TPANI		

Use the FB operator to assign the value of one of the current feedback devices to a variable or to make a comparison. Depending on the configuration of the SFB command, the feedback device could be an encoder or an analog input.

If you issue a PSET command, the feedback device position value will be offset by the PSET command value.

If scaling is **not** enabled, the position values returned will be encoder or ANI counts. If scaling is enabled (SCALE1), the encoder and ANI values will be scaled by the SCLD value. For more information on scaling, refer to page 16.

Syntax: VARn=aFB where n is the variable number, and a is the axis number, or FB can be used in an expression such as IF(1FB<6). An axis specifier must precede the FB operator, or it will default to axis 1 (e.g., VAR1=1FB, IF(1FB<20000), etc.).

Example:

```
SFB1           ; Feedback for axis 1 is encoder #1
VAR6=1FB      ; Assign position (scalable) of encoder #1 (axis 1) to variable #6
IF(1FB<500)   ; If position (scalable) of encoder #1 (axis 1) is less than 500,
              ; do the commands following the IF statement until the NIF command
VAR4=1FB+1000 ; Set variable #4 equal to current position of encoder plus 1,000
NIF           ; End of IF statement
```

FFILT		Following Filter	
Type	Following	Product	Rev
Syntax	<!><@><a>FFILT<i>,<i>,<i>,<i>,<i>,<i>,<i>,<i>	6K	5.0
Units	i = filtering level		
Range	i = 0, 1, 2, 3, or 4		
Default	0		
Response	FFILT *FFILT0,0,0,0,0,0,0,0 1FFILT *1FFILT0		
See Also	FMAXA, FMAXV, FPPEN		

The FFILT command specifies the bandwidth of the low pass filter applied to the measurements of master position. This command is to be used in these situations:

- Measurement of master position is contaminated by either electrical noise (when analog input is the master) or mechanical vibration.
- Measurement noise is minimal, but the motion that occurs on the master input is oscillatory. In this case, using the filter can prevent the oscillatory signal from propagating into the follower axis (i.e., ensuring smoother motion on the follower axis).

The table below shows how the value of the FFILT command specifies the low pass filter's bandwidth:

FFILT Setting	Low pass Filter Bandwidth
0	∞ (no filtering) – <i>default setting</i>
1	120 Hz
2	80 Hz
3	50 Hz
4	20 Hz

Example:

```
FFILT1,2      ; Set filtering bandwidth to 120 Hz for axis 1, and 80 Hz for axis 2
```

FGADV Following Geared Advance

Type	Following	Product	Rev
Syntax	<!><@><a>FGADV<r>, <r>, <r>, <r>, <r>, <r>, <r>	6K	5.0
Units	r = advance distance (scalable)		
Range	0.00000-999,999,999 (scalable with SCLD)		
Default	n/a		
Response	n/a		
See Also	FOLMD, FOLRD, FOLRN, [FS], FSHFD, GOWHEN, SCLD, TFS		

The FGADV command provides the ability to super-impose an advance or retard on Following motion. This is the same ability provided by the FSHFD command, except that the super-imposed motion is also geared to master motion. The FGADV command has the positive or negative “advance” distance as a parameter, but it initiates motion instead of simply setting up the distance. The shape of the super-imposed profile is determined by the FOLMD, FOLRN, and FOLRD commands (just as a normal preset Following move).

The FGADV command profile may be delayed with the GOWHEN command.

A FGADV move may be performed only while the conditions below exist (Following status bit #23, reported with the FS, TFS, and TFSF commands, indicates that it is “OK to do FGADV move”):

- Master is specified with a FOLMAS command
- Following is enabled with the FOLEN command
- The follower axis is either not moving, or moving at constant ratio in continuous mode (MC1)

A FGADV move may not be performed:

- During a preset (MC0) move
- In a compiled profile or program

Following Status (FS, TFS, and TFSF) bit #24 reports if a “FGADV move is underway”.

Example:

```
COMEXC1      ; All command processing during motion
FOLRN25      ; Set numerator of follower-to-master Following ratio
FOLRD10      ; Set denominator of follower-to-master Following ratio
FOLMD1000    ; Set master distance to 1000 units
MC1          ; Enable continuous positioning mode
D+           ; Set direction to positive
FOLEN1       ; Enable Following
GO           ; Ramp up to a 2.5 to 1 ratio over 1000 master distance units
FOLMD500     ; Set master distance to 500 units
FOLRN13      ; Superimposed ratio will be 1.3 (added to 2.5 = 3.8 total)
WAIT(FS.23=B1) ; Wait for OK to do geared advance
              ; (in this case, ramp is complete)
FGAVD400     ; Advance the follower axis 400 counts over a distance
              ; of 500 master counts
WAIT (FS.23=B1) ; Wait for OK to do geared advance
              ; (in this case, FGADV400 super-imposed profile is complete)
FGADV-400    ; Retard the follower axis 400 counts over a distance of
              ; 500 master counts (2.5 - 1.3 = 1.2 net ratio)
```

FMAXA Follower Axis Maximum Acceleration

Type	Following	Product	Rev
Syntax	<!><@><a>FMAXA<r>, <r>, <r>, <r>, <r>, <r>, <r>, <r>	6K	5.0
Units	r = units/sec/sec		
Range	r = 0.00001 - 39,999,998 (scalable with SCLA)		(applicable only to stepper axes)
Default	0.00 (no limit imposed)		
Response	FMAXA *FMAXA0.0000,0.0000,0.0000,0.0000 ... 1FMAXA *FMAXA0.0000		
See Also	FFILT, FMAXV, FPPEN, SCLA		

The FMAXA command sets the maximum acceleration for follower axes. The FMAXA command is scaled by the SCLA parameter.

As part of a ramp to new ratio, or simply following an accelerating master at constant ratio, a follower may be required to accelerate. If the required acceleration is larger than FMAXA, the follower will begin falling behind its commanded position. The 6K controller will attempt to make up this position error as soon as the commanded accel falls below FMAXA. In stepper controllers, an error correction velocity is added to that implied by the commanded ratio.

As with FMAXV, FMAXA should be determined and defined early in the development stage of an application to prevent any damage to the load on the follower axis when unexpectedly high accelerations are commanded. The torque available from the follower motor will also be a determining factor in this parameter in order to prevent motor stalls.

Example:

```
FMAXA75,100 ;Set axis 1 maximum follower acceleration to 75 user units and axis 2  
; maximum acceleration to 100 user units.
```

FMAXV Follower Axis Maximum Velocity

Type	Following	Product	Rev
Syntax	<!><@><a>FMAXV<r>, <r>, <r>, <r>, <r>, <r>, <r>, <r>	6K	5.0
Units	r = units/sec (scalable with SCLV)		
Range	r = 0.000000-1600000.000000		(applicable only to stepper axes)
Default	0.00 (no limit imposed)		
Response	FMAXV *FMAXV0.0000,0.0000,0.0000,0.0000 ... 1FMAXV *FMAXV0.0000		
See Also	FFILT, FMAXA, FPPEN, SCLV		

The FMAXV command sets the maximum velocity at which follower axes may travel. The FMAXV command accepts numeric variables (VAR) as an argument and is scaled by the SCLV parameter.

Normally in a Following application, the follower velocities will be known based on the normal speed of the master and the commanded Following ratios (FOLRN and FOLRD). In some cases, however, the master speed may be higher than normal, the follower may be commanded to perform a shift move, or some other event may occur which will cause the follower to travel at a velocity higher than expected. In these cases, the 6K controller will increase the speed of the follower as necessary to perform the required move, but only up to the FMAXV value.

If the commanded speed is higher than FMAXV, the follower axis will start falling behind its commanded position. The 6K controller will attempt to make up this position error as soon as the commanded speed falls below FMAXV. In stepper controllers, an error correction velocity is automatically added to that implied by the commanded ratio.

The FMAXV value should be determined and defined early in the development stage of an application to prevent any damage to the load on the follower axis when unexpectedly high velocities are commanded.

Example:

```
FMAXV15,20 ;Set the axis 1 follower maximum velocity to 15 user units and  
; axis 2 follower maximum velocity to 20 user units.
```

FMCLLEN Master Cycle Length

Type	Following	Product	Rev
Syntax	<!><@><a>FMCLLEN<r>,<r>,<r>,<r>,<r>,<r>,<r>,<r>	6K	5.0
Units	r = master distance units (scalable)		
Range	r = 0-999,999,999 (scalable with SCLMAS)		
Default	0		
Response	FMCLLEN *FMCLLEN0,0,0,0,0,0,0,0 1FMCLLEN *1FMCLLEN0		
See Also	FMCNEW, FMCP, FOLEN, [FS], GOWHEN, [PMAS], SCLMAS, TFS, TPMAS, WAIT		

The FMCLLEN command defines the length of the master cycle in user units. This value is scaled by the SCLMAS parameter. Numeric variables (VAR) can be used with this command. The initial value for FMCLLEN is zero (FMCLLENØ), which means that the default master cycle length is the maximum internal size (4,294,967,296).

The concept of a master cycle may be useful when moves or other events must be initiated at certain master positions in a repetitive cycle. By specifying a master cycle length, periodic actions may be programmed in a loop or with subroutines which refer to cycle positions, even if the master runs continuously. It is possible to program the 6K controller to suspend program operation or delay moves until specified master cycle positions. The master cycle length, FMCLLEN, should be defined before the functions which wait for periodic master cycle positions are used. An axis need not be in Following mode (FOLEN1) to utilize the concept of a master cycle. However, **master positions will not be measured until a master has been assigned with the FOLMAS command.**

Example (refer also to FOLEN example #2):

```
SCLMAS4000,16000 ; Set the master scale factors: axis 1 = 4000; axis 2 = 16000
FMCLLEN3,(VAR2) ; Set axis 1 master cycle length to 3 user units, and axis 2
                  ; to the value of variable 2 times the SCLMAS value
```

FMCNEW Restart Master Cycle Counting

Type	Following	Product	Rev
Syntax	<!><@><a>FMCNEW	6K	5.0
Units	n/a		
Range	b = 0 (do not restart), 1 (restart immediately), or X (don't change)		
Default	n/a		
Response	n/a		
See Also	FMCLLEN, FMCP, GOWHEN, [NMCY], [PMAS], TPMAS, TRGFN, WAIT		

The FMCNEW1 command restarts master cycle counting. This sets the master cycle position (PMAS) to the value most recently specified with FMCP, and sets the master cycle number (NMCY) to zero. The master cycle position and the master cycle number are set immediately, and program flow continues normally.

The function of the FMCNEW1 command can be initiated with a trigger input by specifying a TRGFNCx1 command. If the FMCNEW1 command is used, master cycle counting is restarted immediately, if TRGFNCx1 is used, the 6K controller will record the instruction to set the master cycle position when the specified trigger occurs. In this case, the master cycle counting is restarted when the specified trigger is activated, even though commands continue to execute and the master cycle counting continues.

FMCNEWØ or FMCNEW1 will remove the status of master cycle restart pending a trigger input (TRGFNCx1). In the case of FMCNEWØ, no restart will occur, and the specified trigger will not cause a new cycle restart. Furthermore, if there is a trigger-based restart pending on axis X, and on axis Y a GOWHEN condition is specified based on PMAS of axis X, then issuing an FMCNEWØ on axis X will clear the pending trigger on axis X and will also clear the pending GOWHEN on axis Y.

A new cycle automatically occurs (i.e., the master cycle position is set to zero, not the FMCP value), when the master cycle length (FMCLLEN) is reached, even if no FMCNEW command has been executed.

Example:

```
TPMAS                ; Display master position: response is *TPMAS12.2,0.5
FMCNEW11             ; Start new master cycle for axes 1 and 2
TPMAS                ; Display master position: response is *0,0
```

FMCP Initial Master Cycle Position

Type	Following	Product	Rev
Syntax	<!><@><a>FMCP<r>, <r>, <r>, <r>, <r>, <r>, <r>	6K	5.0
Units	r = master position in scalable steps		
Range	r = ±999,999,999 (scalable with SCLMAS)		
Default	0		
Response	FMCP *FMCP+0,+0,+0,+0,+0,+0,+0 1FMCP *1FMCP0		
See Also	FMCNEW, FOLMAS, [FS], GOWHEN, SCLMAS, TFS, WAIT		

The FMCP command defines the initial master cycle position in user units. The initial master cycle position is assigned as the current master cycle position each time master cycle counting is restarted with the FMCNEW or TRGFNCx1 command. This value is scaled by the SCLMAS parameter. Numeric variables (VAR) can be used with this command. The default value for FMCP is zero (FMCP0), which means that the master cycle position will be zero when master cycle counting is restarted (see FMCNEW).

The concept of an initial master cycle position may be useful if a new master cycle position counting must be restarted at a master position which is different from what needs to be considered the “zero position” of a periodic cycle. The initial position defined with FMCP applies to the first cycle only. When a master cycle is complete, the master cycle position rolls over to zero. A negative value would be used if some master travel were desired before master cycle position was zero. A positive value would be used if it was necessary to enter the first master cycle at a position greater than zero.

For example, suppose FMCLEN was set to 20 and FMCP was set to 7. When master cycle position counting is restarted, either via FMCNEW1 or the specified trigger (TRGFNCx1), the initial master cycle position will be 7. Rollover will occur after the master travels 13 more units, and the master cycle position would go to zero.

Example:

```
FMCP-2,7          ; Set the initial master cycle position to -2 for axis 1  
                  ; and to 7 for axis 2
```

FOLEN Following Mode Enable

Type	Following	Product	Rev
Syntax	<!><@><a>FOLEN	6K	5.0
Units	n/a		
Range	b = 0 (disable), 1 (enable) or X (don't change)		
Default	0		
Response	FOLEN: *FOLEN0000_0000 1FOLEN: *1FOLEN0		
See Also	FGADV, FOLK, FOLMAS, FOLRD, FOLRN, FOLRNF, [FS], FSHFC, FSHFD, GOWHEN, JOG, JOY, TFS		

The FOLEN command indicates whether subsequent moves on the specified axes will be following a master (FOLEN1) or normal time-based moves (FOLEN0). The term *Following mode* means that FOLEN1 has been given, and that the motion of the follower is dependent on the motion of the master at all times. If FOLEN0 is given, the motion of the master is still monitored, but the motion of the follower is independent of the master.

To move in the Following mode, the master must be previously specified with the FOLMAS command.

Enabling the Following mode (FOLEN1) will set the net position shift value (reported by TPSHF and PSHF) to zero. This is true even if the follower is already in Following mode.

S-Curve profiling is not operational during Following moves.

RESTRICTIONS ON USING FOLEN

The FOLEN command may not be executed during certain conditions (results in the error message “NOT VALID DURING RAMP”).

- You may not enable Following (FOLEN1) on an axis that is in motion, waiting for a GOWHEN condition, or operating in the Joystick mode (JOY1) or Jog mode (JOG1).
 - You may not disable Following (FOLEN0) on an axis that is in motion (unless moving at ratio in continuous mode, MCL, and not shifting) or waiting for a GOWHEN condition.
-

FOLEN Examples

Example #1:

The 6K product is controlling a rotary drive, the master is a 1000-line incremental encoder mounted on the back of an externally controlled motor, and programming units are to be revs/second (rps).

Stepper Products:

The follower will start ramping to a ratio of 1:1 when trigger #1 (TRG-1A) goes active. This means the actual step ratio of follower to master is 25000 to 4000, or 6.25 follower steps for every master. After 25 master revolutions, the follower will decelerate to a 0.5:1 ratio (3.125 follower steps for every master). After a total of 75 master revolutions, the follower will ramp to zero ratio (i.e., stop) and repeat the cycle when trigger #1 is activated. All ramps to new ratios, including zero ratio, take place over one master revolution.

Scaling Set Up: (prior to defining program)

```
SCALE1           ; Enable scaling
SCLD25000        ; Set follower distance scale factor to 25,000 steps/rev
                  ; (assumes a motor/drive res of 25,000 steps/rev)
SCLMAS4000       ; Set master scale factor to 4000 steps/rev
```

Servo Products:

The follower will start ramping to a ratio of 1:1 when trigger #1 (TRG-1A) goes active. This means the actual step ratio of follower to master is 4000 to 4000, or 1 follower steps for every master. After 25 master revolutions, the follower will decelerate to a 0.5:1 ratio (0.5 follower steps for every master). After a total of 75 master revolutions, the follower will ramp to zero ratio (i.e., stop) and repeat the cycle when trigger #1 is activated. All ramps to new ratios, including zero ratio, take place over one master revolution.

Scaling Set Up: (prior to defining program)

```
SCALE1           ; Enable scaling
SCLD4000         ; Set follower distance scale factor to 4,000 steps/rev
                  ; (assumes an encoder resolution of 4,000 steps/rev)
SCLMAS4000       ; Set master scale factor to 4000 steps/rev
```

The application program is defined as follows:

```
DEL FOLTST       ; Delete program called FOLTST
DEF FOLTST       ; Begin definition of program called FOLTST
INFNC1-H        ; Set input #1 (TRG-1A) to be "trigger interrupt" (used with GOWHEN later)
COMEXC1         ; Select continuous command processing mode
MC1             ; Select continuous positioning mode
FOLMAS31        ; Assign encoder input #3 as master for axis #1
FOLMD1          ; Follower should change ratios over 1 master revolution
FMCLN100        ; Set master cycle length to 100 revs
FOLRD1          ; Set follower-to-master Following ratio denominator to 1
                  ; (applies to all subsequent FOLRN commands)
FOLEN1          ; Enable Following on axis #1
D+              ; Set motion to the positive- direction
$STRMV          ; Label to repeat move
1TRGFNA1        ; Suspend execution of next move until trigger (TRG-1A) is active
1TRGFNAx1       ; Begin new master cycle (counter at 0) when trigger (TRG-1A) is active
FOLRN1          ; Set follower-to-master Following ratio numerator to 1 (ratio set to 1:1)
GO1             ; Start continuous Following move (when TRG-1A is active)
WAIT(1AS.26=b0 AND FS.4=b1) ; Wait for profile to actually start
                  ; (when TRG-1A is active) and be at ratio
GOWHEN(1PMAS>=25) ; Suspend execution of next move until master position >= 25
FOLRN0.5        ; Set Following ratio numerator to 0.5 (ratio set to 0.5:1)
GO1             ; Initiate new move according to new Following ratio
                  ; (when master position >= 25)
WAIT(1AS.26=b0 AND FS.4=b1) ; Wait for profile to actually start
                  ; (when master position >= 25) and be at ratio
GOWHEN(1PMAS>=75) ; Suspend execution of next move until master position >= 75
FOLRN0          ; Set Following ratio numerator to zero
                  ; (ratio causes follower to ramp to stop)
GO1             ; Initiate new move with new Following ratio (when master pos. >= 75)
WAIT(1AS.26=b0 AND FS.1=b0) ; Wait for profile to actually start
                  ; (when master position >= 75) and the follower is not moving
JUMP STRMV      ; Repeat the cycle
END             ; End of program
```

Example #2:

Stepper Axes:

The master is an encoder mounted to gearing on a conveyor line. The gearing results in 16,000 encoder steps per conveyor inch. The follower on axis one is a 25,000 step/rev microstepper on a 36" long, 4-pitch leadscrew. The follower waits for the product to be sensed on the conveyor, accelerates to a 1-to-1 ratio, waits for a safe location to actuate the stamping equipment, then applies an inked stamp to the product at the correct location. After the stamp is placed, the follower quickly moves back to the starting position and waits for the next product. Note that this example illustrates how the WAIT command can be used to wait for master cycle positions in order to coordinate motion.

Scaling Set Up: (prior to defining program)

```
SCALE1          ; Enable scaling
SCLA100000      ; Set accel scaling: 100,000 steps/inch
SCLV100000      ; Set velocity scaling: 100,000 steps/inch
SCLD100000      ; Set follower distance scaling: 100,000 steps/inch
SCLMAS16000     ; Set master scale factor to 16000 steps/inch to program in inches
```

Servo Axes:

The master is an encoder mounted to gearing on a conveyor line. The gearing results in 16,000 encoder steps per conveyor inch. The follower on axis one is a 4,000 step/rev servo on a 36" long, 4-pitch leadscrew. The follower waits for the product to be sensed on the conveyor, accelerates to a 1-to-1 ratio, waits for a safe location to actuate the stamping equipment, then applies an inked stamp to the product at the correct location. After the stamp is placed, the follower quickly moves back to the starting position and waits for the next product. Note that this example illustrates how the WAIT command can be used to wait for master cycle positions in order to coordinate motion.

Scaling Set Up: (prior to defining program)

```
SCALE1          ; Enable scaling
SCLA16000       ; Set accel scaling: 16,000 steps/inch
SCLV16000       ; Set velocity scaling: 16,000 steps/inch
SCLD16000       ; Set follower distance scaling: 16,000 steps/inch
SCLMAS16000     ; Set master scale factor to 16,000 steps/inch to program in inches
```

The application program is defined as follows:

```
DEF STAMPR      ; Begin definition of program called STAMPR
COMEXS1        ; Continue command execution after Stop
COMEXC1        ; Continue command execution during motion
SCALE1         ; Enable parameter scaling
1OUTFNC1-A     ; Configure onboard output #1 as a general-purpose prog. output
1INFNC2-H      ; Define TRG-1B as trigger interrupt (use as GOWHEN input)
A10            ; Acceleration = 10 inches/sec/sec
V5             ; Velocity = 5 inches/sec (non-Following moves)
MA1           ; Enable absolute positioning mode for axis 1
FOLMAS21       ; Assign encoder input #2 as master for axis 1
FOLRN1        ; Set follower-to-master Following ratio numerator to 1
FOLRD1        ; Set follower-to-master Following ratio denominator to 1 (ratio is 1:1)
FOLMD1        ; Accel the follower over 1 master inch for Following moves
FMCLN40       ; Master cycle length is 40 inches
$INKON        ; Label to repeat inking process
FOLEN1        ; Enable Following on axis #1
1TRGFNBx1     ; Begin new master cycle when TRG-1B goes active
               ; (product sensed on conveyor)
1TRGFNB1      ; Start next move when TRG-1B is active
D+            ; Set to positive-direction
MC1           ; Select continuous positioning mode
GO1           ; Start continuous follower move on trigger #2
WAIT(1PMAS>=10.5) ; Wait until master position is 10.5 inches - this is when the
               ; stamping device can be actuated without mechanical damage
               ; to the leadscrew assembly
1OUT.1-1      ; Turn on actuator (output #1) to place ink stamp on product
T.1           ; Wait for the ink stamp to be pressed in place by a
               ; stationary stamper
1OUT.1-0      ; Turn off actuator (output #1)
S1            ; Stop follower move
WAIT(1AS.1=b0) ; Wait until the axis is not moving
FOLEN0        ; Disable Following on axis #1
D0            ; Set distance (position) to zero
MCO          ; Select preset positioning mode
GO1           ; Move back to zero (the home position)
WAIT(MOV=b0)  ; Wait until the axis is not moving
JUMP INKON    ; Begin cycle again on trigger #2
END           ; End of program
```

FOLK

Following Kill

Type	Following	Product	Rev
Syntax	<! >FOLK	6K	5.0
Units	n/a		
Range	b= 0 (disable) or 1 (enable)		
Default	0		
Response	FOLK *FOLK0000_0000		
See Also	DRIVE, [ER], ERROR, FOLEN, FOLRD, FOLRN, FOLMAS, FOLMD, FSHFC, FSHFD, INFNC, K, [PSHF], SMPER, TER		

Under default operation (FOLK0), certain error conditions (i.e., drive fault input active, or max. position error limit exceeded) will cause the 6K controller to disable the drive and kill the Following profile (follower's commanded position loses synchronization with the master).

If you enable Following Kill (FOLK1), these error conditions will still disable the drive (DRIVE0), but will not kill the Following profile. Because the Following profile is still running, the controller keeps track of what the follower's position should be in the Following trajectory. To resume Following operation, resolve the error condition (drive fault, excessive position error), enable the drive (DRIVE1), and command the controller to impose a shift to compensate for the lapse/shift that occurred while the drive was disabled and the follower was not moving. To impose the shift, assign the negative of the internally monitored shift value (PSHF) to a variable (e.g., VAR1 = -1 * PSHF) and command the shift using a variable substitution in the FSHFD command (e.g., FSHFD(VAR1)).

The FOLK command only preserves Following profiles; normal velocity-based profiles will be killed regardless of the FOLK command.

FOLMAS

Assignment of Master to Follower

Type	Following	Product	Rev
Syntax	<! >@<a>FOLMAS<±i>, <±i>, <±i>, <±i>, <±i>, <±i>, <±i>, <±i>	6K	5.0
Units	1st i = master axis #; 2nd i = master count source; ± sets direction of master counts relative to direction of actual master count source		
Range	1st i = 1-8 (axis); 2nd i = 1 (encoder), 2 (analog input), 4 (commanded position) 5 (internal count source), or 6 (internal sine wave source). NOTE: "1", by itself, selects the master encoder. "0", by itself, disables the axis from being a follower		
Default	+0 (disable from being a follower axis)		
Response	FOLMAS *FOLMAS+0,+0,+0,+0, +0,+0,+0,+0 1FOLMAS *1FOLMAS+0		
See Also	ANIMAS, FGADV, FOLEN, FOLK, FOLMD, FOLRD, FOLRN, FOLRNF, [FS], FVMACC, FVMFRQ, SINAMP, SINANG, SINGO, TFS		

Use FOLMAS to assign or un-assign a master to a follower axis. Each data field (±i) configures that axis as a follower following the specified master count source. In the syntax for each follower axis (±i), the sign bit sets the direction of master counting relative to the actual direction of the counts as received from the master count source. The first i selects the axis number of the master you are assigning to the follower, and the second i selects the count source of that master axis.

Exceptions to the syntax:

- If a one (1) is placed in the data field (±i), that axis will follow the counts from the Master Encoder (the separate encoder labeled "MASTER ENCODER").
- If a zero (0) is placed in the data field (±i), that axis becomes a normal non-Following axis.

Virtual Master. There are two “Virtual Master” options (an internal count source and an internal sine wave) for applications that require the synchronization features of Following, but have no external master. For a detailed description virtual master features, see “Virtual Master” in the *Programmer’s Guide*.

- Master Source Option 5 (e.g., FOLMAS±i5) selects the internal count source as master. The frequency and acceleration of the internal count source are established with the FVMACC and FVMFRQ commands, respectively.
- Master Source Option 6 (e.g., FOLMAS±i6) selects the internal sine wave as master. The angle and amplitude of the sine wave are established with the SINANG and SINAMP commands, respectively. To start and stop the internal sine wave generator, use the SINGO command.

If scaling is enabled (SCALE1), the measurement of the master is scaled by the SCLMAS value. For more information on scaling, refer to page 16 or to the SCLMAS command description.

NOTES

- A follower axis cannot use its own feedback device or commanded position as the master input.
- Multiple axes may follow the same count source (e.g., encoder) from the same master. However, multiple axes may **not** follow different count sources (e.g., encoder and commanded position) from the same master.
- Before you can use an analog input as a master count source, you must first use the ANIMAS command to assign the analog input to a master axis number. Then you can use the FOLMAS command to assign the analog input as a master counting source for a specific follower axis.

As an example, the FOLMAS+31, -12, , command sets up these parameters:

- Follower axis #1 is set up as follows (+31): Encoder #3 is assigned as the master to follower axis #1. The positive sign bit indicates that master counts will count in the same direction as encoder #3.
- Follower axis #2 is set up as follows (-12): Master analog input #1 is assigned as the master to follower axis #2. The negative sign bit indicates that the master counts will count in the opposite direction of the sign of the voltage change on the analog input.
- Axes 3 and 4 are not affected.

NOTE

The FOLMAS command configures an axis to be a follower, but *does not* automatically enable Following. To enable Following use the FOLEN1 command. To enable follower motion, enable Following (FOLEN1), issue a ratio (FOLRN and FOLRD), and issue the GO command.

As soon as the master is specified with the FOLMAS command, a continuously updated relationship is maintained between the follower’s position and the master’s position. Also, master velocity is continuously measured. **For steppers only**, the configuration of the follower axis is used in the implementation of the step output, so several commands need to be executed before FOLMAS; they are DRES, ERES, and PULSE.

Notice that the master axis number does not need to be the same as the follower axis number. (For example, given FOLMAS21, 44, , 31, axis 1 is follower to the encoder input on axis #2, axis #2 is follower to the commanded output of axis #4, axis #3 is not configured as a follower, and axis 4 is follower to the encoder input of axis #3.)

There are several applications in which a minus sign in the FOLMAS command is used. A minus sign should be used whenever the master is moving in the desired positive direction and yet the 6K controller actually perceives the master to be moving in the negative direction. For example, this can occur when the master input device is mounted on the opposite side of a conveyor. Putting a minus sign in front of the master parameter specification in the FOLMAS command causes the incoming master signal to be negated before it is used by the follower. The term *master count* refers to the count after negation, if any.

For preset follower moves, the direction the follower travels depends on the mode of operation (absolute or incremental) and the commanded position. However, once a preset follower move is commanded, it will

only start moving if the master is moving in the positive direction. This is true no matter the commanded direction of the follower move.

For continuous follower moves, the master count direction has a different effect. If the commanded move is positive in direction and the master is counting up, the actual follower travel direction will be positive. If the commanded move is positive in direction and the master is counting down, the actual follower travel direction will be negative. Similar cases exist for follower moves commanded in the negative direction.

Example: (refer to the FOLEN examples)

FOLMD		Master Distance			
Type	Following			Product	Rev
Syntax	<!><@><a>FOLMD<r>,<r>,<r>,<r>,<r>,<r>,<r>,<r>			6K	5.0
Units	i = distance in counts				
Range	0 - 999,999,999 (scalable by SCLMAS)				
Default	0				
Response	FOLMD *FOLMD0,0,0,0,0,0,0,0 1FOLMD *FOLMD0				
See Also	ANIMAS, FGADV, FOLEN, FOLK, FOLRN, FOLRNF, FOLRD, MC, [PMAS], SCLMAS, TPMAS				

If a follower is in continuous positioning mode (MC1), FOLMD is the master distance over which acceleration or deceleration from the current ratio to the new ratio takes place. Or, if a follower is in preset positioning mode (MCØ), the FOLMD command indicates the master distance over which the next preset move will take place.

If scaling is enabled (SCALE1), the FOLMD value is specified in user units and is scaled by the SCLMAS parameter (for more detail on scaling, refer to page 16 or to the SCLMAS command description). Numeric variables (VAR) can be used with this command (e.g., FOLMD12, (VAR6), 3, 6).

By carefully specifying accurate master distances for each ramp of a follower's move profile, a precise position relationship between master and follower will be maintained during all phases of the profile. The "Master and Follower Distance Calculation" section in the Following chapter of the *Programmer's Guide* discusses the relationship between ratio changes and the corresponding master and follower distances.

HINT: If a follower is in continuous mode (MC1) and the master is starting from rest, setting FOLMD to Ø will ensure precise tracking of the master's acceleration ramp. This is how the trackball application example is written in the Following chapter of the *Programmer's Guide*.

Examples: (refer also to FOLEN example #2)

```

SCALE1          ; Enable parameter scaling
SCLMAS4000      ; Master scale factor is 4000 steps/rev
SCLD4000        ; Follower scale factor is 4000 steps/rev
DEL progx       ; Delete program called progx
DEF progx       ; Begin definition of program called progx
FOLMAS31        ; Axis 3 encoder is the master for axis 1
FOLMD0          ; Assign Following acceleration distance to 0 master revs
                ; (i.e., instantaneous)
FOLRN1          ; Set follower-to-master Following ratio numerator to 1
FOLRD1          ; Set follower-to-master Following ratio denominator to 1
                ; Ratio set to 1:1
FOLEN1          ; Enable Following on axis #1
D-              ; Set direction to opposite direction of the master
GO1             ; Begin following master. If the master is not moving, follower
                ; will remain at rest until master moves, at which time the
                ; follower will track the master precisely, but in the opposite
                ; direction as the master.
END             ; End definition of progx

```

FOLRD Denominator of Follower-to-Master Ratio

Type	Following	Product	Rev
Syntax	<!><@><a>FOLRD<r>, <r>, <r>, <r>, <r>, <r>, <r>, <r>	6K	5.0
Units	r = master distance in counts		
Range	r = 1.00000 - 999,999,999 (scalable by SCLMAS)		
Default	1		
Response	FOLRD *FOLRD1,1,1,1,1,1,1,1 1FOLRD *FOLRD1		
See Also	COMEXC, FGADV, FOLEN, FOLK, FOLMAS, FOLRN, FOLRNF, SCLMAS		

The FOLRD command establishes the denominator of a ratio between follower and master travel. (Ratios are always specified as positive, similar to velocity.) For a preset move (MCØ), it is the maximum allowed ratio, and for a continuous move (MC1), it is the final ratio reached by the follower. The actual follower direction will depend on commanded moves (D+ or D-) and master direction.

If no FOLRD parameter is specified, it is assumed to be 1.

If scaling is enabled (SCALE1), the FOLRD value is scaled by the SCLMAS value. For more detail on scaling, refer to page 16 or to the SCLMAS command description.

Numeric variables (VAR) can be used with this command for master parameters (e.g., FOLRD(VAR5) , 5).

Each time FOLRN or FOLRD are given, the 6K controller divides the scaled numerator and denominator to calculate the ratio, but roundoff errors are eliminated by measuring both master and follower over a large distance. After scaling, the maximum magnitude of the ratio is 127 follower steps for every master step.

ON-THE-FLY CHANGES: You can change Following ratio *on the fly* (while motion is in progress) in two ways. One way is to send an immediate command (!FOLRD) followed by an immediate go command (!GO). The other way is to enable the continuous command execution mode (COMEXC1) and execute a buffered command (FOLRD) followed by a buffered go command (GO).

Example: (refer also to the FOLEN examples)

```
SCLD25000      ; Set follower scaling factor to 25,000
SCLMAS4000     ; Set master scaling factor to 4,000
SCALE1        ; Enable scaling
FOLRN5         ; Set ratio numerator to 5 (5 * 25,000 = 125,000)
FOLRD3         ; Set ratio denominator to 3 (3 * 4,000 = 12,000)
               ; (Resulting ratio is 125 follower steps to every 12 master steps.)
```

FOLRN Numerator of Follower-to-Master Ratio

Type	Following	Product	Rev
Syntax	<!><@><a>FOLRN<r>, <r>, <r>, <r>, <r>, <r>, <r>, <r>	6K	5.0
Units	r = follower distance in steps		
Range	r = 0.00000 - 999,999,999.99999 (scalable by SCLD)		
Default	1		
Response	FOLRN *FOLRN1,1,1,1,1,1,1,1 1FOLRN *FOLRN1		
See Also	FGADV, FOLEN, FOLK, FOLMAS, FOLRNF, FOLRD, SCLD		

The FOLRN command establishes the numerator of a ratio between follower and master travel. (Ratios are always specified as positive, similar to velocity.) For a preset move (MCØ), it is the maximum allowed ratio, and for a continuous move (MC1), it is the final ratio reached by the follower. The actual follower direction will depend on commanded moves (D+ or D-) and master direction.

If no FOLRN parameter is specified, it is assumed to be 1.

If scaling is enabled (SCALE1), the FOLRN value is scaled by the SCLD value. For more detail on scaling, refer to page 16 or to the SCLD command description.

Numeric variables (VAR) can be used with this command for follower parameters (e.g., FOLRN(VAR2) , 5).

Each time FOLRN or FOLRD are given, the 6K controller divides the scaled numerator and denominator to calculate the ratio, but roundoff errors are eliminated by measuring both master and follower over a large distance. After scaling, the maximum magnitude of the ratio is 127 follower steps for every master step.

ON-THE-FLY CHANGES: You can change Following ratio *on the fly* (while motion is in progress) in two ways. One way is to send an immediate command (!FOLRN) followed by an immediate go command (!GO). The other way is to enable the continuous command execution mode (COMEXC1) and execute a buffered command (FOLRN) followed by a buffered go command (GO).

Example: refer to the FOLRD and FOLEN examples

FOLRNF	Numerator of Final Follower-to-Master Ratio, Preset Moves	Product	Rev
Type	Following; Compiled Motion	6K	5.0
Syntax	<!><@><a>FOLRNF<r>,<r>,<r>,<r>,<r>,<r>,<r>,<r>		
Units	r = follower distance in steps		
Range	0.00000		
Default	0		
Response	FOLRNF *FOLRNF0,0,0,0,0,0,0,0 !FOLRNF *!FOLRNF0		
See Also	FGADV, FOLEN, FOLRD, FOLRN, FOLMD, SCLD		

The Numerator of Final Follower-to-Master Ratio, Preset Moves (FOLRNF) command establishes the numerator of the final ratio between follower and master travel. (Ratios are always specified as positive, similar to velocity.) The FOLRNF command designates that the motor will move the load the distance designated in a preset GOBUF segment, completing the move at a final ratio of zero. FOLRNF applies only to the first subsequent GOBUF, which marks an intermediate “end of move” within a following profile. FOLRNF is used only in conjunction with the GOBUF command. Normal preset GO moves always finish with zero FOLRNF.

If scaling is enabled (SCALE1), the FOLRNF value is scaled by the SCLD value. For more detail on scaling, refer to page 16 or to the SCLD command description.

NOTE: The only allowable value for FOLRNF is 0, and it may only be used with compiled preset Following moves (a non-zero FOLRNF value will result in an immediate error message). FOLRNF is allowed for a segment only if the starting ratio is also zero (i.e., it must be the first segment, or the previous segment must have ended in zero ratio).

With compiled preset Following moves where FOLRNF has not been given, the final ratio is given with FOLRN, and the shape of the intermediate profile will be constrained to be within the starting and ending ratios.

For more information on using the FOLRNF command, refer to the Custom Profiling chapter in the *Programmer's Guide*.

FPPEN	Master Position Prediction Enable	Product	Rev
Type	Following	6K	5.0
Syntax	<!><@><a>FPPEN		
Units	n/a		
Range	b = 0 (disable), 1 (enable) or X (don't change)		
Default	1		
Response	FPPEN *FPPEN1111_1111 !FPPEN *!FPPEN1		
See Also	[FS], TFS		

The FPPEN command enables or disables Master Position Prediction in the 6K controller Following algorithm. Master Position Prediction is enabled by default, but can be disabled as desired with the FPPEN0 command.

The 6K controller measures master position once per *position sample period* and calculates a corresponding follower commanded position. This calculation, and achieving the subsequent follower commanded position, requires 2 sample periods (4 milliseconds).

Enabling Master Position Prediction (FPPEN1) eliminates any lag in follower position which would be dependent on master speed. It may be desirable to disable Master Position Prediction (FPPEN0) when maximum follower smoothness is important and minor phase delays can be accommodated. A detailed discussion of Master Position Prediction is given in the Following chapter of the *Programmer's Guide*.

Example:
FPPEN1 ; Enable Master Position Prediction for axis 1 and 2.

[FS]**Following Status**

Type	Following; Assignment or Comparison	Product	Rev
Syntax	See below	6K	5.0
Units	n/a		
Range	n/a		
Default	n/a		
Response	n/a		
See Also	FGADV, FMCLLEN, FMCP, FOLEN, FOLMAS, FPPEN, FSHFC, FSHFD, MC, [NMCY], [PMAS], TFS, TFSF, VARB		

The Following Status (FS) command is used to assign the Following status bits for a specific axis to a binary variable, or to make a comparison against a binary or hexadecimal value. The function of each status bit is shown below.

Bit Assignment (left to right)	Function (YES = 1; NO = 0)	
1	Follower in Ratio Move	A Following move is in progress.
2	Ratio is Negative	The current ratio is negative (i.e., the follower counts are counting in the opposite direction from the master counts).
3	Follower Ratio Changing	The follower is ramping from one ratio to another (including a ramp to or from zero ratio).
4	Follower At Ratio	The follower is at constant non-zero ratio.
* 5	FOLMAS Active	A master is specified with the FOLMAS command.
* 6	FOLEN Active	Following has been enabled with the FOLEN command.
* 7	Master is Moving	The specified master is currently in motion.
8	Master Dir Neg	The current master direction is negative. (bit must be cleared to allow Following move in preset mode=MC0).
9	OK to Shift	Conditions are valid to issue shift commands (FSHFD or FSHFC).
10	Shifting now	A shift move is in progress.
11	Shift is Continuous	An FSHFC-based shift move is in progress.
12	Shift Dir is Neg	The direction of the shift move in progress is negative.
13	Master Cyc Trig Pend	A master cycle restart is pending the occurrence of the specified trigger.
14	Mas Cyc Len Given	A non-zero master cycle length has been specified with the FMCLLEN command.
15	Master Cyc Pos Neg	The current master cycle position (PMAS) is negative. This could be by caused by a negative initial master cycle position (FMCP), or if the master is moving in the negative direction.
16	Master Cyc Num > 0	The master position (PMAS) has exceeded the master cycle length (FMCLLEN) at least once, causing the master cycle number (NMCY) to increment.
17	Mas Pos Prediction On	Master position prediction has been enabled (FPPEN).
18	Mas Filtering On	A non-zero value for master position filtering (FFILT) is in effect.
19	RESERVED	
20	RESERVED	
21	RESERVED	
22	RESERVED	
23	OK to do FGADV move	OK to do Geared Advance move (master assigned with FOLMAS, Following enabled with FOLEN, and follower axis is either not moving, or moving at constant ratio in continuous mode).
24	FGADV move underway	Geared Advance move profile is in progress.

* All these conditions must be true before Following motion will occur.

Syntax: VARBn=aFS where n is the binary variable number and a is the axis identifier, or FS can be used in an expression such as IF(1FS=b1101), or IF(1FS=h7F). The FS command must be used with an axis specifier, or it will default to axis 1.

To make a comparison against a binary value, place the letter b (b or B) in front of the value that the Following status is being compared against. The binary value itself must only contain ones, zeros, or Xs (1, 0, X, x). To make a comparison against a hexadecimal value, the letter h (h or H) must be placed in front of the value that the Following status is being compared against. The hexadecimal value itself must only contain the letters A through F, and the numbers 0 through 9.

If you wish to assign only one bit of the Following status to a binary variable, instead of all 32, the bit select (.) operator can be used. The bit select, in conjunction with the bit number, is used to specify a specific Following status bit (e.g., VARB1=1FS.12 assigns axis 1 status bit 12 to binary variable 1).

Example:

```

VARB1=1FS      ; Following status for axis 1 assigned to binary variable 1
VARB2=1FS.12   ; Axis 1 Following status bit 12 assigned to binary variable 2
VARB2          ; Response if bit 12 is set to 1 should be:
               ; *VARB2=XXXX XXXX XXX1 XXXX XXXX XXXX XXXX XXXX
IF(4FS=b111011X11) ; If the Following status for axis 4 contains 1's for
                 ; inputs 1, 2, 3, 5, 6, 8, and 9, and a 0 for bit location 4,
                 ; do the IF statement
TREV          ; Transfer revision level
NIF          ; End if statement
IF(2FS=h7F00) ; If the Following status for axis 2 contains 1's for inputs 1,
                 ; 2, 3, 5, 6, 7, and 8, and 0's for every other bit location,
                 ; do the IF statement
TREV          ; Transfer revision level
NIF          ; End if statement

```

FSHFC Continuous Shift

Type	Following	Product	Rev
Syntax	<!><@><a>FSHFC<i>,<i>,<i>,<i>,<i>,<i>,<i>,<i>,<i>	6K	5.0
Units	i = shift feature to implement		
Range	i = 0 (stop), 1 (positive-direction), 2 (negative-direction), or 3 (kill)		
Default	n/a		
Response	n/a		
See Also	FGADV, FOLEN, FOLK, FOLRN, FOLRNF, FOLRD, [FS], FSHFD, MC, [PSHF], TFS, TPSHF		

The FSHFC command allows time-based follower moves to be superimposed on continuous Following moves. This results in a *shift* (change in phase) between the master position and the follower position. Continuous shift moves in the positive- or negative-direction may be commanded only while the follower is in the Following mode (FOLEN1).

Steppers only: An FSHFC move may be performed only when the follower is in the continuous positioning mode (MC1) and performing a Following move at a constant ratio.

The most recently commanded velocity (V) and acceleration (A) for the follower axis will determine the speed at which the FSHFC move takes place. The velocity and direction of the FSHFC shift is independently superimposed on whatever velocity and direction results from the ratio and motion of the master. The FSHFC shift is not a change in ratio; rather, it is a velocity added to a ratio. The velocity commanded is added to the present speed at which the follower is moving, up to the velocity limit of the product. For example, assume a follower is traveling at 1 rps in the positive direction while following a master. If a FSHFC move is commanded in the positive direction at 2 rps, the follower's actual velocity (after acceleration) will be 3 rps.

The FSHFC parameters stop (0) and kill (3) can be used to halt a continuous FSHFC move (positive-direction or negative-direction). The example below shows how to stop a FSHFC continuous move.

An FSHFC move may be needed to adjust the relative follower position on the fly during the continuous Following move. For example, suppose an operator is visually inspecting the follower's motion with respect

to the master. If he notices that the master and follower are out of synchronization, it may be desirable to have an interrupt programmed (e.g., activated with a push-button switch) that will allow the operator to advance or retard the follower at a super-imposed correction speed until the operator chooses to have the follower start tracking the master again. The example below shows this.

FSHFC Example:

Assume all scale factors and set-up parameters have been entered for the master and follower. In this example, the follower (axis #1) is continually following the master at a 1:1 ratio. If the operator notices some mis-alignment between master and follower, he can press 1 of 2 pushbuttons (connected to onboard trigger inputs #1 and #2, which are also referred to as TRG-1A and TRG-1B) to shift the follower in the positive- or negative-direction at 0.1 user scaled units until the button is released. After the adjustment, the program continues on as before.

Example:

```

DEF SHIFT          ; Begin definition of program called SHIFT
V.1               ; Add or subtract 0.1 user scaled units from the follower velocity
                 ; when shifting
COMEXS1          ; Continue command execution after stop
COMEXC1          ; Continue command execution during motion
FOLMAS31         ; Axis 3 encoder input is the master for axis 1
FOLRN1           ; Set follower-to-master Following ratio numerator to 1
FOLRD1           ; Set follower-to-master Following ratio denominator to 1
                 ; (ratio set to 1:1)
FOLEN1           ; Enable Following mode on axis #1
D+               ; Set to positive-direction
MC1              ; Select continuous positioning mode
GO1              ; Start following master continuously
VARB1=b10        ; Define onboard input pattern #1 and assign to VARB1
VARB2=b01        ; Define onboard input pattern #2 and assign to VARB2
$TESTIN          ; Define label called TESTIN
IF(IN=VARB1)     ; IF statement (if onboard input #1 is activated, do the jump)
  JUMP SHIFTP    ; Jump to shift follower in the positive-direction when pattern 1
                 ; active
  NIF            ; End of IF statement
IF(IN=VARB2)     ; IF statement (if onboard input #2 is activated, do the jump)
  JUMP SHIFTN    ; Jump to shift follower in the negative-direction when pattern 2
                 ; active
  NIF            ; End of IF statement
JUMP TESTIN      ; Return to main program loop
$SHIFTP          ; Define label called SHIFTP (subroutine to shift in the
                 ; positive direction)
FSHFC1           ; Start continuous follower shift move in positive-direction
WAIT(IN.1=b0)    ; Continue shift until onboard input #1 is deactivated
FSHFCØ           ; Stop shift move
JUMP TESTIN      ; Return to main program loop
$SHIFTN          ; Define label called SHIFTN (subroutine to shift in the
                 ; negative-direction)
FSHFC2           ; Start continuous follower shift move in the negative-direction
WAIT(IN.2=b0)    ; Continue shift until onboard input #2 is deactivated
FSHFCØ           ; Stop shift move
JUMP TESTIN      ; Return to main program loop
END              ; End definition of program called SHIFT

```

FSHFD

Preset Shift

Type	Following	Product	Rev
Syntax	<!><@><a>FSHFD<r>, <r>, <r>, <r>, <i>, <i>, <i>, <i>	6K	5.0
Units	r = shift distance		
Range	r = 0.00000 - 999,999,999 (scalable with SCLD)		
Default	n/a		
Response	n/a		
See Also	FGADV, FOLEN, FOLK, FOLRN, FOLRNF, FOLRD, [FS], FSHFC, MC, ONCOND, [PSHF], SCLD, TFS, TPSHF		

The FSHFD command allows time-based follower moves to be superimposed on continuous Following moves. This results in a *shift* (change in phase, or registration) between the master position and the follower position. Preset shift moves of defined or variable distances, may be commanded only while the follower is in the Following mode (FOLEN1). The FSHFD distance is scaled by the SCLD value of scaling is enabled (SCALE1).

Steppers Only: An FSHFD move may be performed only when the follower is in the continuous positioning mode (MC1) and performing a Following move at a constant ratio.

The most recently commanded velocity (V) and acceleration (A) for the follower axis will determine the speed at which the FSHFD move takes place. The velocity and direction of the FSHFD shift is independently superimposed on whatever velocity and direction results from the ratio and motion of the master.

The FSHFC parameters stop (Ø) and kill (3) can be used to halt an FSHFD.

It should be noted that FSHFD is similar in execution to GO. The entire preset distance shift, or ramp-to-shift velocity, must finish before the 6K controller proceeds to the next command.

The FSHFD shift is not a change in ratio; rather, it is a velocity added to a ratio. The velocity commanded will be added to the present speed at which the follower is moving, up to the velocity limit of the product. For example, assume a follower is traveling at 1 rps in the positive direction while following a master. If a FSHFD move is commanded in the positive direction at 2 rps, the follower's actual velocity (after acceleration) will be 3 rps.

An FSHFD move may be needed to adjust the follower position on the fly because of a load condition which changes during the continuous Following move. For example, suppose an operator is visually inspecting the follower's motion with respect to the master. If the operator notices that the master and follower are out of synchronization, it may be desirable to have an input programmed (e.g., activated with a push-button switch) that will allow the operator to advance or retard the follower a fixed distance, and then let the follower resume tracking the master. The example below illustrates this.

FSHFD Example:

Assume all scale factors and set-up parameters have been entered for the master and follower. In this example, the follower (axis #1) is continually following the master at a 1:1 ratio. If the operator notices some mis-alignment between master and follower, he can press 1 of 2 pushbuttons (connected to onboard trigger inputs #1 and #2, which are also referred to as TRG-1A and TRG-1B) to advance or retard the follower a fixed distance of 200 steps. After the adjustment, the follower resumes tracking the master as before.

(Program on following page)

Example:

```

DEF PSHIFT          ; Begin definition of program called PSHIFT
COMEXS1            ; Continue command execution after stop
COMEXC1            ; Continue command execution during motion
FOLMAS31           ; Axis 3 encoder input is the master for axis 1
FOLRN1             ; Set follower-to-master Following ratio numerator to 1
FOLRD1             ; Set follower-to-master Following ratio denominator to 1
                   ; (ratio set to 1:1)
FOLEN1             ; Enable Following mode on axis #1
D+                 ; Set direction to positive
MC1                ; Select continuous positioning mode
GO1                ; Start following master continuously
VARB1=b10          ; Define input pattern #1 and assign to VARB
VARB2=b01          ; Define input pattern #2 and assign to VARB
$TESTIN            ; Define label called TESTIN
IF(IN=VARB1)       ; IF statement (if onboard input #1 is activated, do the jump)
  JUMP SHIFTP      ; Jump to shift follower in positive-direction when pattern 1 active
  NIF              ; End of IF statement
IF(IN=VARB2)       ; IF statement (if onboard input #2 is activated, do the jump)
  JUMP SHIFTN      ; Jump to shift follower in negative-direction when pattern 2 active
  NIF              ; End of IF statement
JUMP TESTIN        ; Return to main program loop
$SHIFTP            ; Define label called SHIFTP (subroutine to shift in the
                   ; positive direction)
FSHFD200           ; Start preset follower shift move of 200 steps in positive direction
WAIT(FS.10=b0)    ; Wait for shift to finish
JUMP TESTIN        ; Return to main program loop
$SHIFTN            ; Define label called SHIFTN (subroutine to shift in the
                   ; negative direction)
FSHFD-200          ; Start preset follower shift move of 200 steps in the negative
                   ; direction
WAIT(FS.10=b0)    ; Wait for shift to finish
JUMP TESTIN        ; Return to main program loop
END                ; End definition of program called PSHIFT

```

FVMACC Virtual Master Count Acceleration

Type	Following	Product	Rev
Syntax	<!><@><a>FVMACC<i>,<i>,<i>,<i>,<i>,<i>,<i>,<i>	6K	1.0
Units	i = count acceleration in counts/sec/sec		
Range	±999,999,999.9999		
Default	+0		
Response	FVMACC *FVMACC+0,+0,+0,+0,+0,+0,+0,+0 1FVMACC *1FVMACC+0		
See Also	FOLMAS, FVMFRQ, SINAMP, SINANG, SINGO		

Use the FVMACC command to define the rate at which the virtual master internal count frequency may change for each axis. This command allows smooth changes in master velocity and direction.

FVMFRQ Virtual Master Count Frequency

Type	Following	Product	Rev
Syntax	<!><@><a> FVMFRQ<i>,<i>,<i>,<i>,<i>,<i>,<i>,<i>	6K	1.0
Units	i = count frequency in counts/sec		
Range	±1000000.0000		
Default	+0		
Response	FVMFRQ *FVMFRQ+0,+0,+0,+0,+0,+0,+0 1FVMFRQ *1FVMFRQ+0		
See Also	FOLMAS, FVMACC, SINAMP, SINANG, SINGO		

Use the FVMFRQ command to define the virtual master count frequency for each axis. The “virtual master” is an internal count source, intended to mimic the counts which might be received on an external encoder port. Just as may be encountered with an external encoder, this count source may speed up, slow down, stop, or count backwards.

There is one count source per axis. Each count source has a variable count frequency, defined by the user. The count sources are always enabled, counting at the signed rate specified by this command. To start and stop the count source, specify non-zero or zero values, respectively, for the FVMFRQ command.

The rate at which the count frequency may change is specified in counts per second per second with the FVMACC command, allowing smooth changes in master velocity and direction.

GO Initiate Motion

Type	Motion	Product	Rev
Syntax	<!><@>GO	6K	5.0
Units	n/a		
Range	b = 0 (don't go), 1 (go), or X (don't change)		
Default	1		
Response	GO: No response; instead, motion is initiated on all axes		
See Also	A, AA, AD, ADA, COMEXC, D, DRFLVL, GOBUF, GOWHEN, K, LH, LS, MA, MC, PSET, S, SCLA, SCLD, SCLV, SSV, TEST, V		

The Initiate Motion (GO) command instructs the motor to make a move using motion parameters that have been previously entered. Several commands affect the motion that will occur when a GO is received: SCLA, SCLD, SCLV, A, AA, AD, ADA, D, V, LH, LS, MA, and MC.

The GO command starts motion on any or all axes. If the GO command is issued without any arguments, motion will be started on all axes.

If motion does not occur after a GO command has been issued, verify the drive fault level (DRFLVL) and the limits (LH and LS).

On-The-Fly (Pre-emptive GO) Motion Profiling

While motion is in progress (regardless of the positioning mode), you can change these motion parameters to affect a new profile:

- Acceleration (A) — S-curve acceleration is not supported in OTF motion changes
- Deceleration (AD) — S-curve acceleration is not supported in OTF motion changes
- Velocity (V)
- Distance (D)
- Preset or Continuous Positioning Mode Selection (MC)
- Incremental or Absolute Positioning Mode Selection (MA)
- Following Ratio Numerator and Denominator (FOLRN and FOLRD, respectively)

The motion parameters can be changed by sending the respective command (e.g., A, V, D, MC, etc.) followed by the GO command. If the continuous command execution mode is enabled (COMEXC1), you can execute buffered commands; otherwise, you must prefix each command with an immediate command identifier (e.g., !A, !V, !D, !MC, etc., followed by !GO). The new GO command pre-empts the motion profile in progress with a new profile based on the new motion parameter(s).

For more information, refer to the Custom Profiling section in the *Programmer's Guide*.

Example:

```
SCALE1           ; Enable scaling
SCLA25000,25000,1,1 ; Set the accel. scale factor on axes 1 & 2 to
                  ; 25000 steps/unit, axes 3 & 4 to 1 step/unit
SCLV25000,25000,1,1 ; Set the velocity scale factor on axes 1 & 2 to
                  ; 25000 steps/unit, axes 3 & 4 to 1 step/unit
SCLD1,1,1,1      ; Set the distance scaling factor on axes 1, 2, 3, & 4 to
                  ; 1 step/unit
DEL proga        ; Delete program called proga
DEF proga        ; Begin definition of program called proga
MA0000           ; Incremental positioning mode on all axes
MC0000           ; Preset positioning mode on all axes
A10,12,1,2      ; Set the acceleration to 10, 12, 1, & 2 units/sec/sec
                  ; on axes 1, 2, 3 & 4
V1,1,1,2        ; Set the velocity to 1, 1, 1, & 2 units/sec on
                  ; axes 1, 2, 3 & 4
D100000,1000,10,100 ; Set the distance to 100000, 1000, 10, & 100 units on
                  ; axes 1, 2, 3 & 4
GO1100          ; Initiate motion on axes 1 and 2, 3 & 4 do not move
END             ; End definition of proga
```

GOBUF Store a Motion Segment in Compiled Memory

Type	Compiled Motion	Product	Rev
Syntax	<@>GOBUF	6K	5.0
Units	n/a		
Range	b = 0 (don't go), 1 (go), or X (don't change)		
Default	1		
Response	n/a		
See Also	[AS], DEF, END, [ER], FOLRNF, MA, MC, MEMORY, PCOMP, PEXE, POUTn, PRUN, PUCOMP, PLOOP, PLN, [SS], TAS, TER, TSS, VF		

The Store a Motion Segment in Compiled Memory (GOBUF) command creates a motion segment as part of a profile and places it in a segment of compiled memory, to be executed after all previous GOBUF motion segments have been executed. When a GOBUF command is executed, the distance from the new D command is added to the profile's current goal position as soon as the GOBUF command is executed, thus extending the overall move distance of the profile under construction.

GOBUF is not a stand-alone command; it can only be executed within compiled programs, using the PCOMP and PRUN commands.

Each GOBUF motion segment may have its own distance to travel, velocity, acceleration and deceleration. The end of a preset segment (MC0) is determined by the distance or position specified; a compiled MC0 GOBUF motion segment is finished when the "D" goal is reached. The end of a continuous segment (MC1) is determined by the ratio or velocity specified; a compiled MC1 GOBUF motion segment is finished when the

velocity or ratio goal is reached. If either a preset segment or continuous segment is followed by a compiled GOWHEN command, motion will continue at the last velocity until the GOWHEN condition becomes true, and the next segment begins.

The GOBUF command is not allowed during absolute positioning mode (MA1).

Starting velocity of a GOBUF segment

Every GOBUF motion segment will start at a velocity equal to the previous segment's end velocity. If the previous GOBUF segment uses the VFØ command, then it will end at zero velocity; otherwise, the end velocity will equal to the goal velocity (V) of the previous segment.

Ending velocity of a GOBUF segment

Preset Positioning Mode (MCØ)

A preset motion segment starts at the previous motion segment's end velocity, attempts to reach the goal velocity (V) with the programmed acceleration and deceleration (A and AD) values, and is considered completed when the distance (D) goal is reached.

In non-Following motion (FOLENØ), the last preset GOBUF segment always ends at zero velocity, but if you wish the velocity between intermediate GOBUF segments to end at zero velocity, use the VFØ command. In Following mode (FOLEN1), the last preset GOBUF segment will end with the last-specified goal velocity, but if you wish the velocity between intermediate GOBUF segments to end at zero velocity, use the FOLRNF command.

Each GOBUF will build a motion segment that, by default, becomes known as the last segment in the profile. The last motion segment in a profile must end at zero velocity. If using pre-compiled loops (PLOOP) and the loop is closed after the last GOBUF segment (PLN occurs after the last GOBUF), then the unit will not consider the last GOBUF as a final motion segment since it can link to either the first segment of the loop or the next segment after the loop. If the conditions are such that the last motion segment is within a loop and does not end at zero velocity, then an error is generated (TSS/SS bit #31 is set) at compile time (PCOMP), and the profile remains un-compiled.

Continuous Positioning Mode (MC1)

A continuous segment starts at the previous motion segment's end velocity, and is considered complete when it reaches the goal velocity (V) at the programmed accel (A) or decel (AD) values.

You may use a mode continuous (MC1) non-zero velocity segment as the last motion segment in a profile (no error will result). The axis will just continue traveling at the goal velocity.

NOTE: Each GOBUF motion segment can consume from 2-8 memory segments of compiled memory. If there is no more space left in compiled memory, a compilation error will result.

Example:

```

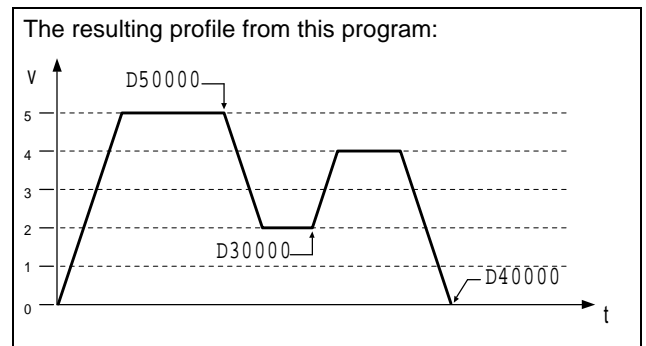
DEF simple ; Begin definition of program
MC0 ; Preset positioning mode
MA0 ; Preset incremental
; positioning mode
D50000 ; Distance is 50000
A10 ; Acceleration is 10
AD10 ; Deceleration is 10
V5 ; Velocity is 5
GOBUF1 ; 1st motion segment, axis 1
D30000 ; Distance is 30000
V2 ; Velocity is 2
GOBUF1 ; 2nd motion segment, axis 1
D40000 ; Distance is 40000
V4 ; Velocity is 4
GOBUF1 ; 3rd motion segment, axis 1
END ; End program definition

```

```

PCOMP simple ; Compile simple
PRUN simple ; Run simple

```



GOL Initiate Linear Interpolated Motion

Type	Motion (Linear Interpolated)	Product	Rev
Syntax	<!><@>GOL	6K	5.0
Units	n/a		
Range	b = 0 (don't go), 1 (go), or X (don't change)		
Default	0		
Response	GOL: No response, instead motion is initiated on all axes		
See Also	D, GOWHEN, PA, PAA, PAD, PADA, PV, SCALE, SCLA, SCLD, SCLV		

The Initiate Linear Interpolated Motion (GOL) command instructs the motor to make a move using motion parameters that have been previously entered. Several commands affect the motion that will occur when a GOL is received: PA, PAA, PAD, PADA, D, PV, and SCLA, SCLD, SCLV.

The GOL command starts motion on any or all axes. If the GOL command is issued without any arguments, motion will be started on all axes.

When moves are made using the GOL command, the endpoint of the linear interpolated move is determined by the D command. The accelerations, decelerations, and velocities for the individual axes are calculated internally by the 6K Series product, so that the load is moved *in a straight line* at the path acceleration (PA and PAD) and velocity entered (PV). In other words, the path acceleration (PA), path average acceleration (PAA), the path deceleration (PAD), path average deceleration (PADA), and the path velocity (PV) all correspond to the rate of travel required to go to the point in space specified by the D command. All axes are to arrive at the same time; therefore, if each axis' distance is different, each axis must travel at a different rate to have each axis arrive at the same time. The 6K Series product takes care of the calculations for each axis, you just enter the overall rate of travel.

If motion does not occur after a GOL command has been issued, verify the drive fault level (DRFLVL) and the limits (LH and LS).

Example:

```
SCALE1          ; Enable scaling
@SCLA25000      ; Set path acceleration scale factor to 25000 steps/unit
@SCLV25000      ; Set path velocity scale factor to 25000 steps/unit
@SCLD10000      ; Set distance scale factor to 10000 steps/unit on all axes
DEL conta      ; Delete program called conta
DEF conta      ; Begin definition of program called conta
PA25            ; Set the path acceleration to 25 units/sec/sec
PAD20           ; Set the path deceleration to 20 units/sec/sec
PV2            ; Set the path velocity to 2 units/sec
D10,5,2,11     ; Set the distance to 10, 5, 2, and 11 units on axes 1-4
GOL1111        ; Initiate linear interpolated motion on axes 1-4. A GOL command
                ; could have been issued instead of a GOL1111 command.
END            ; End definition of conta
```

GOSUB Call a Subroutine

Type	Program; Subroutine Definition; Program Flow Control	Product	Rev
Syntax	<!>GOSUB<t>	6K	5.0
Units	t = text (name of program/subroutine)		
Range	Text name of 6 characters or less		
Default	n/a		
Response	n/a		
See Also	\$, BREAK, DEF, DEL, END, ERASE, GOTO, JUMP, RUN		

The Call a Subroutine (GOSUB) command branches to the corresponding program/subroutine name when executed. A subroutine name consists of 6 or fewer alpha-numeric characters. The subroutine that the GOSUB initiates will return control to the line after the GOSUB, when the subroutine completes operation. If an invalid subroutine name is entered, no branch will occur, and processing will continue with the line after the GOSUB.

If you do not want to use the GOSUB command before the subroutine name (GOSUBsubname), you can simply use the subroutine name without the GOSUB attached to it (subname).

If a subroutine is executed, and a BREAK command is received, the subroutine will return control to the calling program or subroutine immediately.

Up to 16 levels of subroutine calls can be made without receiving an error.

Example:

```

DEF pick          ; Begin definition of subroutine named pick
GO1100           ; Initiate motion on axes 1 and 2
END              ; End subroutine definition
DEF place        ; Begin definition of subroutine named place
GOSUB pick       ; Gosub to subroutine named pick
GO1000           ; Initiate motion on axis 1
END              ; End subroutine definition
place           ; Execute program named place

```

After program `place` is initiated, the first thing to occur will be a `gosub` to program `pick`. Within `pick`, the `GO` command will be executed, and then control will be passed back to program `place`. The `GO` command in `place` will then be executed, and program execution will then terminate.

GOTO**Goto a Program or Label**

Type	Program; Subroutine Definition; Program Flow Control	Product	Rev
Syntax	<!>GOTO<t>	6K	5.0
Units	t = text (name of program/label)		
Range	Text name of 6 characters or less		
Default	n/a		
Response	n/a		
See Also	\$, DEF, DEL, END, GOSUB, IF, JUMP, L, LN, NIF, NWHILE, REPEAT, RUN, UNTIL, WHILE		

The `GOTO` command branches to the corresponding program name or label when executed. A program or label name consists of 6 or fewer alpha-numeric characters. The program or label that the `GOTO` initiates will **not** return control to the line after the `GOTO` when the program completes operation—instead, the program will end. This holds true unless the subroutine in which the `GOTO` resides was called by another program; in this case, the `END` in the `GOTO` program will initiate a return to the calling program.

If an invalid program or label name is entered, the `GOTO` will be ignored, and processing will continue with the line after the `GOTO`.

CAUTION

Use caution when performing a `GOTO` between `IF & NIF`, or `L & LN`, or `REPEAT & UNTIL`, or `WHILE & NWHILE`. Branching to a different location within the same program will cause the next `IF`, `L`, `REPEAT` or `WHILE` statement to be nested within the previous `IF`, `L`, `REPEAT` or `WHILE` statement unless a `NIF`, `LN`, `UNTIL` or `NWHILE` command has already been encountered. If you wish to avoid this nesting situation, use the `JUMP` command instead of the `GOTO` command.

Example:

```

DEF pick          ; Begin definition of subroutine named pick
GO1100           ; Initiate motion on axes 1 and 2
END              ; End subroutine definition
DEF place        ; Begin definition of subroutine named place
GOTO pick        ; Goto to subroutine named pick
GO1000           ; Initiate motion on axis 1
END              ; End subroutine definition
place           ; Execute program named place
; After the GOTO command, the GO1000 command will not be executed because a GOTO
; was issued. If a GOSUB was used instead of the GOTO statement, control would
; have been returned to the line after the GOSUB.

```

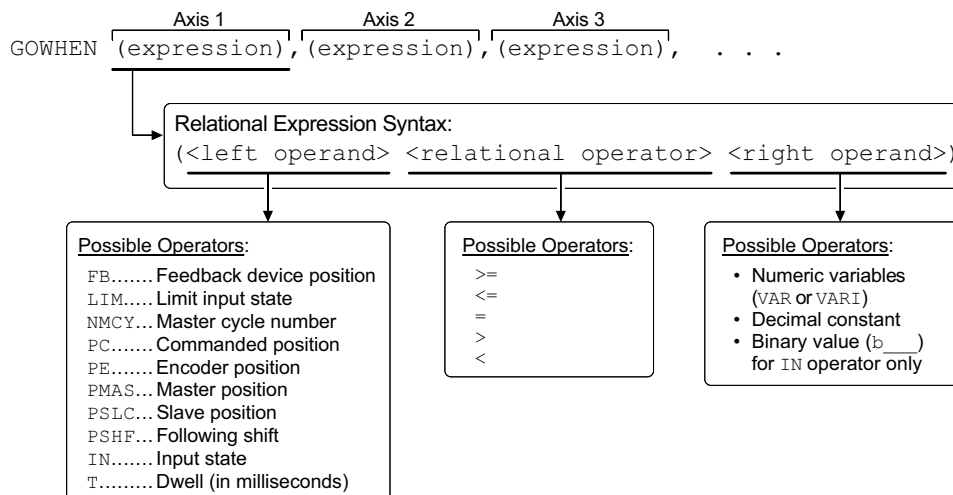
GOWHEN Conditional Go

Type	Motion; Following	Product	Rev
Syntax	<!><@><a>GOWHEN(expression,expression,...) (1 expression per axis -- see diagram below)	6K	5.0
Units	n/a		
Range	Up to 80 characters (including parentheses)		
Default	n/a		
Response	n/a		
See Also	[AS], COMEXC, [ER], ERROR, ERRORP, [FB], FGADV, FSHFC, FSHFD, GO, GOL, [IN], [LIM], [NMCY], [PC], [PE], [PMAS], [PSHF], [PSLV], T, TAS, TER, TRGFN, WAIT		

The GOWHEN command is used to synchronize a motion profile of an axis with a specified position count (commanded, feedback device, motor, master, follower, Following shift), input status, dwell (time delay), or master cycle number on that axis or other axes. Command processing does not wait for the GOWHEN conditions (relational expressions) to become true during the GOWHEN command. Rather, the motion from the subsequent start-motion command (GO, GOL, FGADV, FSHFC, and FSHFD) will be suspended until the condition becomes true.

Start-motion type commands that **cannot** be synchronized using the GOWHEN command are: HOM, JOG, JOY, and PRUN. A preset GO command that is already in motion can start a new profile using the GOWHEN and GO sequence of commands. Continuous moves (MCL) already in progress can change to a new velocity based upon the GOWHEN and GO sequence. Both preset and continuous moves can be started from rest with the GOWHEN and GO sequence.

GOWHEN Syntax:



EXAMPLES

```
GOWHEN (1PE>40000) ; suspend next GO until axis 1 encoder position > 40000
GOWHEN (IN.6=b1) ; suspend next GO until onboard input #6 is activated (b1)
GOWHEN (2PMAS>255) ; suspend next GO until the master for axis 2 has
; traveled 255 master distance units
```

SCALING

If scaling is enabled (SCALE1), the right-hand operand is multiplied by SCLD if the left-hand operand is FB, PC, PE, PSLV, or PSHF. The right-hand operand is multiplied by the SCLMAS value if the left-hand operand is PMAS. The SCLD or SCLMAS values used correlate to the axis specified with the variable (e.g., a GOWHEN expression with 3PE scales the encoder position by the SCLD value specified for axis 3).

GOWHEN Status:

Axis Status — Bit #26: Bit #26 is set when motion has been commanded by a GO, GOL, FGADV, FSHFC, or FSHFD command, but the change in motion is suspended due to a pending GOWHEN condition. This status bit is cleared when the GOWHEN condition is true or when a stop (!S) or kill (!K or ^K) command is executed. An individual axis' GOWHEN command can be cleared using an axis-specific S or K command (e.g., !S11XØ or !KØXX1).

AS . 26 Assignment & comparison operator — use in a conditional expression (see AS).
TASF Full text description of each status bit. (see “Gowhen is Pending” line item)
TAS Binary report of each status bit (bits 1-32 from left to right). [See bit #26.](#)

Error Status — Bit #14: Bit #14 is set if the position relationship specified in the GOWHEN command is already true when the GO, GOL, FGADV, FSHFC, or FSHFD command is issued. The error status is monitored and reported only if you enable error-checking bit #14 with the ERROR command (e.g., ERROR . 14-1). NOTE: When the error occurs, the controller will branch to the error program (assigned with the ERRORP command).

ER . 14 Assignment & comparison operator — use in a conditional expression (see AS).
TERF Full text description of each status bit. (see “Gowhen condition true” line item)
TER Binary report of each status bit (bits 1-32 from left to right). [See bit #14.](#)

GOWHEN ... On a Trigger Input:

If you wish motion to be triggered with a trigger input, use the aTRGFNC1 command. The aTRGFNC1 command executes in the same manner as the GOWHEN command, except that motion is executed when the specified trigger input (c) for axis (a) is activated. For more information, refer to the TRGFN command description.

GOWHEN vs. WAIT:

A WAIT will cause the 6K controller program to halt program flow (except for execution of immediate commands) until the condition specified is satisfied. Common uses for this function include delaying subsequent I/O activation until the master has achieved a required position or an object has been sensed.

By contrast, a GOWHEN will suspend the motion profile for a specific axis until the specified condition is met. It does **not** affect program flow. If you wish motion to be triggered with a trigger input, use the aTRGFNC1 command. The aTRGFNC1 command executes in the same manner as the GOWHEN command, except that motion is executed when the specified trigger input (c) is activated (see TRGFN command description for details). In addition, GOWHEN expressions are limited to the operands listed above; WAIT can use additional operands such as FS (Following status) and VMAS (velocity of master).

Factors Affecting GOWHEN Execution:

If, on the same axis, a second GOWHEN command is executed **before** a start-motion command (GO, GOL, FGADV, FSHFC, or FSHFD), then the first GOWHEN is over-written by the second GOWHEN command. (GOWHEN commands are not nested.) An error is not generated when a GOWHEN command is over-written by another GOWHEN.

While waiting for a GOWHEN condition to be met **and** a start-motion command **has** been issued, if a second GOWHEN command is encountered, then the first sequence is disabled and another start-motion command is needed to re-arm the second GOWHEN sequence.

A new GOWHEN command must be issued for each start-motion command (GO, GOL, FGADV, FSHFC, or FSHFD). That is, once a GOWHEN condition is met and the motion command is executed, subsequent motion commands will not be affected by the same GOWHEN command.

If the GOWHEN and start-motion commands are issued, the motion profile is delayed until the GOWHEN condition is met. If a second start-motion command is encountered, the second start-motion command will override the GOWHEN command and start motion. If this override situation is not desired, it can be avoided by using a WAIT condition between the first start-motion command and the second start-motion command.

It is probable that the GOWHEN command, the GO command, and the GOWHEN condition becoming true may be separated in time, and by other commands. Situations may arise, or commands may be given which make the GOWHEN invalid or inappropriate. In these cases, the GOWHEN condition is cleared, and any motion pending the GOWHEN condition becoming true is canceled. These situations include execution of the JOG, JOY, HOM, PRUN, and DRIVEØ commands, as well motion being stopped due to hard or soft limits, a drive fault, an immediate stop (!S), or an immediate kill (!K or ^K).

GOWHEN in Compiled Motion: When used in a compiled program, a GOWHEN will pause the profile in progress (motion continues at constant velocity) until the GOWHEN condition evaluates true. When executing a compiled Following profile, the GOWHEN is ignored on the reverse Following path (i.e., when the master is moving in the opposite direction of that which is specified in the FOLMAS command). A compiled GOWHEN may require up to 4 segments of compiled memory storage.

Sample 6K Code:

In the example below, axis 2 must start motion when the actual position of axis 1 has reached 4. While axis 1 is moving, the program must be monitoring inputs and serving other system requirements, so a WAIT statement cannot be used; instead, a GOWHEN and GO sequence will delay the profile of axis 2.

```
SCALE1          ; Enable scaling
SCLV25000,25000 ; Set velocity scaling factors
SCLD10000,10000 ; Set distance scaling factors
DEL proga       ; Delete program called proga
DEF proga       ; Begin definition of program called proga
MC00            ; Set both axes to preset move mode
D20,20         ; Set distance end-point
COMEXC1        ; Enable continuous command execution mode
V1,1           ; Set velocity
A100,100       ; Set acceleration
GOWHEN(,1PE>4) ; Delay axis 2 profile. When the expression is true
                ; (position of encoder #1 is > 4), allow axis 2 to
                ; start motion.
GO11           ; Command both axes to move. Axis 2 will not start until
                ; conditions in the GOWHEN statement are true.
                ; Command processing does not wait, so other system
                ; functions may be performed.
END            ; End definition of program
```