Automation

6K Series
Command Reference

Effective: November 5, 1998

INFNC OUTFNC C TSKAX GO WHILE
LS HELP OUT IF
TDIR LIMFNC A PS ONIN PCOMP
D LH WHEN ELSE
TRGFN TPE GOBUF SFB
WAIT DEF END PRUN
RESET
STARTP VAR VARB
ERASE DEL
JOY JOG VARS
TASF ENC MC
MA FOLEN TOUT DRFEN
HOM TIO
READ DRES
ERROR TSTAT FOLMAS ERES
STRGTE TSSF
GOWHEN TFSF CMDDIR DATSIZ
TIN GOTO
SOFFS V TREV S K TINOF

IMPORTANT

User Information

! **WARNING** !

6K Series products are used to control electrical and mechanical components of motion control systems. You should test your motion system for safety under all potential conditions. Failure to do so can result in damage to equipment and/or serious injury to personnel.

Motion Planner and Servo Tuner are trademarks of Parker Hannifin Corporation.
Microsoft and MS-DOS are registered trademarks, and Windows, Visual Basic, and Visual C++ are trademarks of Microsoft Corporation.

Technical Assistance ⇨ *Contact your local automation technology center (ATC) or distributor, **or** ...*

North America and Asia:
Compumotor Division of Parker Hannifin
5500 Business Park Drive
Rohnert Park, CA 94928
Telephone: (800) 358-9070 or (707) 584-7558
Fax: (707) 584-3793
FaxBack: (800) 936-6939 or (707) 586-8586
e-mail: tech_help@cmotor.com
Internet: http://www.compumotor.com

Europe (non-German speaking)**:**
Parker Digiplan
21 Balena Close
Poole, Dorset
England BH17 7DX
Telephone: +44 (0)1202 69 9000
Fax: +44 (0)1202 69 5750

Germany, Austria, Switzerland:
HAUSER Elektronik GmbH
Postfach: 77607-1720
Robert-Bosch-Str. 22
D-77656 Offenburg
Telephone: +49 (0)781 509-0
Fax: +49 (0)781 509-176

Parker
Automation

Product Feedback Welcome
E-mail: 6Kuser@cmotor.com

Introduction

Purpose of this Document

This document is designed as a reference for all the 6K Series commands. To gain a full understanding of how the 6K Series commands are used together to implement specific features, refer to the *6K Series Programmer's Guide* (p/n 88-017137-01). For hardware-related information (e.g., electrical wiring connections, specifications, tuning, etc.), refer to the *6K Series Hardware Installation Guide*.

Table of Contents

www.comoso.com

Description of Format

1.	2.		3.

		Product	**Rev**
INEN	**Input Enable**	6K	5.0

4.	Type	Inputs or Program Debug Tools
5.	Syntax	`<!><B>INEN<d><d><d>...<d>`
6.	Units	`d = 0, 1, E, or X`
7.	Range	`0 = off, 1 = on, E = enable, X = don't care`
8.	Default	`E`
9.	Response	`INEN: *INENEEEE_EEEE_EEEE_EEEE_EEEE_EEEE_EEEE`
10.	See Also	`[IN], INFNC, INLVL, INPLC, INSTW, TIN, TIO`

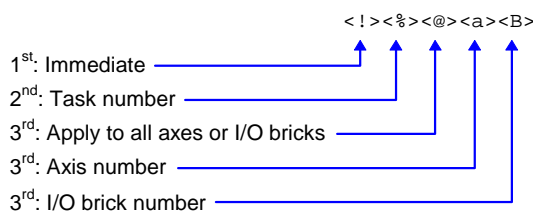| **Item Number**	**Description**
1.	**Mnemonic Code**: This field contains the command's mnemonic code. If the command is in brackets (e.g., `[ IN ]`), it is an operator that must be used within the syntax of another command (e.g., `IN` may be used in a conditional expression like `IF(IN.3=b1)`).
2.	**Full Name**: This field contains the command's full name.
3.	**Valid Product & Revision**: This field lists the 6K Series products and the revision of each product when this command was incorporated or modified per the description. If the command does not apply to that particular product, the **Rev** is specified as "n/a".
	You can use the `TREV` command to determine which product revision you are using. For example, if the `TREV` response is `*TREV92-012222-01-5.0`, the product revision is 5.0.
4.	**Type**: This field contains the command's type. Inside the back cover you will find a list of all 6K Series commands organized by command type.
5.	**Syntax**: The proper syntax for the command is shown here. The specific parameters associated with the command are also shown. Definitions of the parameters are described in the *Syntax* sections below.
6.	**Units**: This field describes what unit of measurement the parameter (`b`, `d`, `i`, `r`, or `t`) in the command syntax represents.
7.	**Range**: This is the range of valid values that you can specify for an argument (or any other parameter specified).
8.	**Default**: The default setting for the command is shown in this field. A command will perform its function with the default setting if you do not provide a value.
9.	**Response**: Some commands allow you to check the status of the command. In the example above, entering the `INEN` command by itself, you will receive the response `*INENEEEE_EEEE_EEEE_EEEE_EEEE_EEEE_EEEE` (response indicates all inputs are enabled). The example responses provided are based on the default error level, Error Level 4, established with the `ERRLVL4` command.
10.	**See Also**: Commands related or similar to the command described are listed here.

Syntax -- Letters and Symbols

The command descriptions provided within this manual use alphabetic letters and ASCII symbols within the **Syntax** description (see example below) to represent different parameter requirements.

INEN   Input Enable

			Product	**Rev**
Type	`Inputs or Program Debug Tools`		6K	5.0
→ Syntax	`<!><%><B>INEN<d><d><d>...<d>`			
Units	`d = 0, 1, E, or X`			
Range	`0 = off, 1 = on, E = enable, X = don't care`			
Default	`E`			
Response	`INEN: *INENEEEE_EEEE_EEEE_EEEE_EEEE_EEEE_EEEE`			
See Also	`[IN], INFNC, INLVL, INPLC, INSTW, TIN, TIO`			

Letter/Symbol	Description
`a`	Represents an axis specifier, numeric value from 1 to 8.
`B`	Represents the number of the product's I/O brick. External I/O bricks are represented by numbers 1 through n (to connect external I/O bricks, refer to your product's *Installation Guide*). On-board I/O are address at brick location zero (Ø). If the brick identifier is omitted from the command, the controller assumes the command is supposed to affect the onboard I/O.
`b` *	Represents the values `1`, `0`, `X` or `x`; does not require field separator between values.
`c`	Represents a character (`A` to `Z`, or `a` to `z`)
`d`	Represents the values `1`, `0`, `X` or `x`, `E` or `e` ; does not require field separator between values. `E` or `e` enables a specific command field. `X` or `x` leaves the specific command field unchanged or ignored. In the `ANIEN` command, the "`d`" symbol may also represent a real numeric value.
`i`	Represents a numeric value that cannot contain a decimal point (integer values only). The numeric range varies by command. Field separator required.
`r`	Represents a numeric value that may contain a decimal point, but is not required to have a decimal point. The numeric range varies by command. Field separator required.
`t`	Represents a string of alpha numeric characters from 1 to 6 characters in length. The string must start with a alpha character.
`!`	Represents an immediate command. Changes a buffered command to an immediate command. Immediate commands are processed immediately, even before previously entered buffered commands.
`%`	(Multitasking Only) Represents a task identifier. To address the command to a specific task, prefix the command with "`i%`", where "`i`" is the task number. For example, the `4%CUT` command uses task #4 to execute the program called "`CUT`".
`,`	Represents a field separator. Commands with the symbol `r` or `i` in their Syntax description require field separators. Commands with the symbol `b` or `d` in their Syntax description **do not** require field separators (but they may be included). See *General Guidelines* table below.
`@`	Represents a global specifier, where only one field need be entered. Applicable to all commands with multiple command fields. (e.g., `@V1` sets velocity on all axes to 1 rps).
`< >`	Indicates that the item contained within the `< >` is optional, not required by that command. NOTE: Do not confuse with `<cr>`, `<sp>`, and `<lf>`, which refer to the ASCII characters corresponding to a carriage return, space, and line feed, respectively.
`[ ]`	Indicates that the command between the `[ ]` must be used in conjunction with another command, and cannot be used by itself.

* The ASCII character `b` can also be used within a command to precede a binary number. When the `b` is used in this context, it is not to be replaced with a `0`, `1`, `X`, or `x`. Examples are assignments such as `VARB1=b10001`, and comparisons such as `IF(3IN=b1001X1)`.

Order of Precedence for Command Prefix Characters (from left to right):

```
<!><%><@><a><B>
```

1st: Immediate

2nd: Task number

3rd: Apply to all axes or I/O bricks

3rd: Axis number

3rd: I/O brick number

Syntax -- General Guidelines

Guideline Topic	Guideline	Examples

Command Delimiters (`<cr>`, `<lf>`, and `:`)	All commands must be separated by a delimiter. A carriage return is the most commonly used. The colon (`:`)allows you to place multiple commands on one line of code.	Set acceleration on axis 2 to 10 rev/sec/sec:
		`A,10,,<cr>`
		`A,10,,<lf>`
		`A,10,,: V,25,, : D,25000,, : @GO<cr>`

Neutral Characters (`<sp>` and `<tab>`)	Using neutral characters anywhere within a command will not affect the command.	Set velocity on axis 1 to 10 rps, axis 2 to 25 rps:
		`V<sp>10,<sp>25,,<cr>`
		
		Add a comment to the command:
		`V 10, 25,,<tab> ;set accel.<cr>`

Case Sensitivity	There is no case sensitivity. Use upper or lower case letters within commands.	Initiate motion on axes 1, 3 and 4:
		`GO1011`
		`go1011`

Comment Delimiter (;)	All text between a comment delimiter and a command delimiter is considered *program comments*.	Add a comment to the command: `V10<tab> ;set velocity`
Field Separator (`,`)	Commands with the symbol `r` or `i` in their Syntax description require field separators.	Set velocity on axes 1 - 4 to 10 rps, 25 rps, 5 rps and 10 rps, respectively: `V10,25,5,10`

	Commands with the symbol `b` or `d` in their Syntax description **do not** require field separators (but they may be included).	Initiate motion on axes 1, 3 and 4:
		`GO1011<cr>`
		`GO1,0,1,1`

	Axes not participating in the command need not be specified; however, field separators that are normally required must be specified (unless the axis prefix is used).	Set velocity on axes 4 and 6 to 5 rps:
		`V,,,5,,5`
		Alternative is to use the axis prefix:
		`4V5,,5`

Global Command Identifier (`@`)	When you wish to set the command value equal on all axes, add the `@` symbol at the beginning of the command (enter only the value for one command field).	Set velocity on all axes to 10 rps: `@V10`
	The `@` symbol is also useful for checking the status of all axes, or all inputs or outputs on all I/O bricks.	Check the status of all digital outputs (onboard, and on external I/O bricks): `@OUT`

Bit Select Operator (`.`)	The bit select operator allows you to affect one or more binary bits without having to enter all the preceding bits in the command.
	Syntax for setup commands:
	`[command name].[bit #]-[binary value]`
	Syntax for conditional expressions:
	`[command name].[bit #]=[binary value]`
		Enable error-checking bit #9:
		`ERROR.9-1`
		Enable error-check bits #9-12:
		`ERROR.9-1,1,1,1`
		IF statement based on value of axis status bit #12 for axis #1:
		`IF(1AS.12=b1)`

Left-to-right Math	All mathematical operations assume left-to-right precedence.	`VAR1=5+3*2` Result: Variable 1 is assigned the value of 16 (8*2), not 11 (5+6).
Binary and Hexadecimal Values	When making assignments with or comparisons against binary or hexadecimal values, you must precede the binary value with the letter "`b`" or "`B`", and the hex value with "`h`" or "`H`". In the binary syntax, an "`x`" simply means the status of that bit is ignored.	Binary: `IF(IN=b1x01)` ↑ Hexadecimal: `IF(IN=h7F)` ↑

Multi-tasking Task Identifier (`%`)	Use the `%` command prefix to identify the command with a specific task.	Launch the "`move1`" program in Task 1:
		`1%move1`
		Check the error status for Task 3:
		`3%TER`
		Check the system status for Task 3:
		`3%TSS`

NOTE: The command line is limited to 80 characters (excluding spaces).

Syntax -- Command Value Substitutions

Many commands can substitute one or more of its command field values with one of these substitution items (demonstrated in the programming example below):

VAR.............Places current value of the numeric variable in the corresponding field of the command.

VARB ....Uses the value of the binary variable to establish <u>all</u> the fields in the command.

VARI ....Places current value of the integer variable in the corresponding field of the command.

READ ....Information is requested at the time the command is executed.

DREAD.........Reads the RP240's numeric keypad into the corresponding field of the command.

DREADF ......Reads the RP240's function keypad into the corresponding field of the command.

TW................Places the current value set on the thumbwheels in the corresponding field of the command.

DAT..............Places the current value of the data program (DATP) in the corresponding field of the command.

Programming Example: *(NOTE: The substitution item must be enclosed in parentheses.)*

```
VAR1=15          ; Set variable 1 to 15
A5,(VAR1),4,4    ; Set acceleration to 5,15,4,4 for axes 1-4, respectively
VARB1=b1101XX1   ; Set binary variable 1 to 1101XX1 (bits 5 & 6 not affected)
GO(VARB1)        ; Initiate motion on axes 1, 2 & 4 (value of binary
                 ; variable 1 makes it equivalent to the GO1101 command)
OUT(VARB1)       ; Turn on outputs 1, 2, 4, and 7
VARS1="Enter Velocity" ; Set string variable 1 to the message "Enter Velocity"
V2,(READ1)       ; Set the velocity to 2 on axis 1. Read in the velocity for
                 ; axis 2 , output variable string 1 as the prompting message
                 ; 1. Operator sees "ENTER VELOCITY" displayed on the screen.
                 ; 2. Operator enters velocity prefixed by !' (e.g., !'20).
HOMV2,1,(TW1)    ; Set the home velocity to 2 and 1 on axes 1 and 2, respectively.
                 ; Read in the home velocity for axis 3 from thumbwheel set 1
HOMV2,1,(DAT1)   ; Set the home velocity to 2 and 1 on axes 1 and 2, respectively.
                 ; Read home velocity for axis 3 from data program 1.
VARI1=2*3        ; Set integer variable 1 to 6 (2 multiplied by 3)
D(VARI2),,(VARI3)  ; Set the distance of axis 1 equal to the value of
                 ; integer variable 2, and the distance of axis 3 equal to
                 ; the value of integer variable 3.
```

Rule of Thumb

Not all of the commands allow command field substitutions. In general, commands with a binary command field (`<b>` in the syntax) will accept the `VARB` substitution. Commands with a real or integer command field (`<r>` or `<i>` in the syntax) will accept `VAR`, `VARI`, `READ`, `DREAD`, `DREADF`, `TW` or `DAT`.

---

www.comoso.com

Programmable I/O Bit Patterns

The 6K product has programmable inputs and outputs. The total number of onboard inputs and outputs (trigger inputs, limit inputs, digital outputs) depends on the product. The total number of expansion inputs and outputs (analog inputs, digital inputs and digital outputs) depends on your configuration of expansion I/O bricks.

These programmable I/O are represented by binary bit patterns, and it is the bit pattern that you reference when programming and checking the status of specific inputs and outputs. The bit pattern is referenced 1 to *n*, from left to right.

- Onboard I/O. For example, the status command to check all onboard trigger inputs is `TIN`.
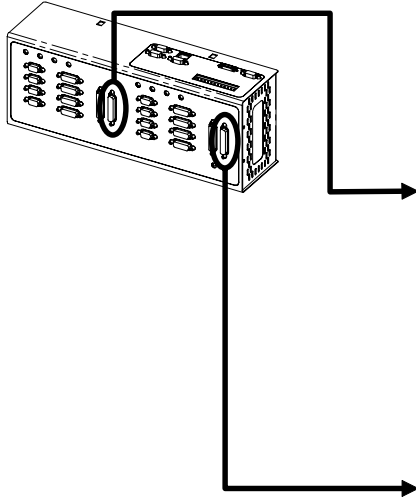 An example response for the 6K8 is: `*TIN0100_0001_0000_0011_0`.
  Bit 1 ⬆   Bit 17 ⬆

- Expansion I/O. For example, the status command to check all digital inputs on I/O brick 2 is `2TIN`.
  An example response for the 6K8 is: `*2TIN0010_0110_1100_0000_XXXX_XXXX_XXXX_XXXX`.
  I/O Brick 2 ⬆   ⬆ Bit 1   Bit 32 ⬆

Onboard I/O

I/O	Location	Programming	Status Report, Assignment
Limit Inputs	"LIMITS/HOME" connectors	`LIMFNC, LIMEN, LIMLVL`	`TLIM, LIM`
Trigger Inputs	"TRIGGERS/OUTPUTS" connectors (pins 9, 11, 13, 15, 17, 19, 21 & 23). Master Trigger is "MASTER TRIG" on connector on top of the 6K chassis	`INFNC, INLVL, INEN, ONIN, INPLC, INSTW`	`TIN, IN`
Outputs (digital)	"TRIGGERS/OUTPUTS" connectors (pins 1, 3, 5 & 7).	`OUT, OUTFNC, OUTLVL, OUTEN, OUTALL, OUTPLC, OUTTW, POUT`	`TOUT, [OUT]`

Limit Inputs ("LIMITS/HOME" connectors)

Input bit pattern for `LIM`, `TLIM`, `LIMEN`, `LIMFNC`, and `LIMLVL`:

Bit #	Pin #	Function *
1	23	Positive end-of-travel limit, axis 1.
2	21	Negative end-of-travel limit, axis 1.
3	19	Home limit, axis 1.
4	17	Positive end-of-travel limit, axis 2.
5	15	Negative end-of-travel limit, axis 2.
6	13	Home limit, axis 2.
7	11	Positive end-of-travel limit, axis 3.
8	9	Negative end-of-travel limit, axis 3.
9	7	Home limit, axis 3.
10	5	Positive end-of-travel limit, axis 4.
11	3	Negative end-of-travel limit, axis 4.
12	1	Home limit, axis 4.
13	23	Positive end-of-travel limit, axis 5.
14	21	Negative end-of-travel limit, axis 5.
15	19	Home limit, axis 5.
16	17	Positive end-of-travel limit, axis 6.
17	15	Negative end-of-travel limit, axis 6.
18	13	Home limit, axis 6.
19	11	Positive end-of-travel limit, axis 7.
20	9	Negative end-of-travel limit, axis 7.
21	7	Home limit, axis 7.
22	5	Positive end-of-travel limit, axis 8.
23	3	Negative end-of-travel limit, axis 8.
24	1	Home limit, axis 8.

* The functions listed are the factory default functions; other functions may be assigned with the `LIMFNC` command.

Sample response to `TLIM` (limit inputs status) command:
`*TLIM001_001_001_001_001_001_001_001`

Trigger Inputs ("TRIGGERS/OUTPUTS" connectors)

Input bit pattern for `TIN`, `IN`, `INFNC`, `INLVL`, `INEN`, `INPLC`, `INSTW`, and `ONIN`:

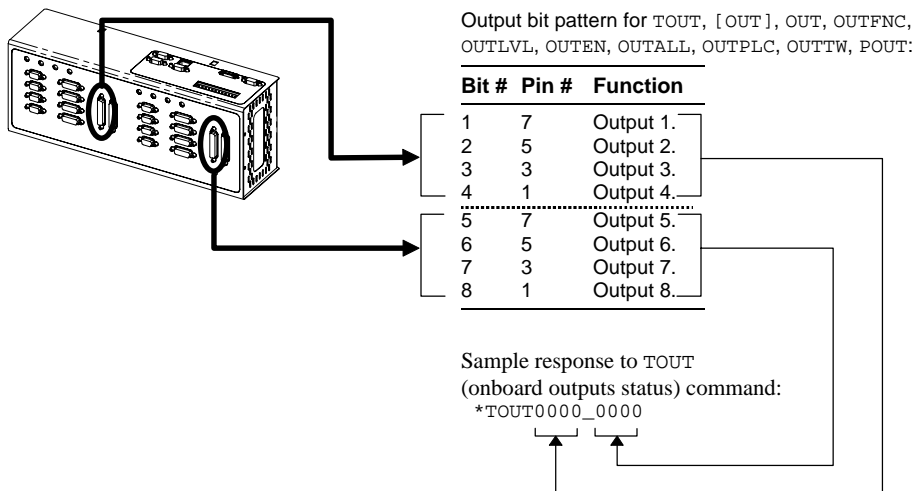| Bit #	Pin #	Function *
1	23	Trigger input 1 (TRIG-1A).
2	21	Trigger input 2 (TRIG-1B).
3	19	Trigger input 3 (TRIG-2A).
4	17	Trigger input 4 (TRIG-2B).
5	15	Trigger input 5 (TRIG-3A).
6	13	Trigger input 6 (TRIG-3B).
7	11	Trigger input 7 (TRIG-4A).
8	9	Trigger input 8 (TRIG-4B).
9	23	Trigger input 9 (TRIG-5A).
10	21	Trigger input 10 (TRIG-5B).
11	19	Trigger input 11 (TRIG-6A).
12	17	Trigger input 12 (TRIG-6B).
13	15	Trigger input 13 (TRIG-7A).
14	13	Trigger input 14 (TRIG-7B).
15	11	Trigger input 15 (TRIG-8A).
16	9	Trigger input 16 (TRIG-8B).
17	----	Trigger input 17 (TRIG-M).

"MASTER TRIG"

* If the input is assigned the "trigger interrupt" function with the `INFNCi-H` command, it will capture the position of the dedicated "*n*" axis identified in the input's name (TRIG-*n*A and TRIG-*n*B). TRIG-M captures the position of the master encoder, as well as all axes.

Sample response to `TIN` (trigger inputs status) command:
```
*TIN0000_0010_1100_0000_0
```

Outputs ("TRIGGERS/OUTPUTS" connectors)

Output bit pattern for `TOUT`, `[OUT]`, `OUT`, `OUTFNC`, `OUTLVL`, `OUTEN`, `OUTALL`, `OUTPLC`, `OUTTW`, `POUT`:

Bit #	Pin #	Function
1	7	Output 1.
2	5	Output 2.
3	3	Output 3.
4	1	Output 4.
5	7	Output 5.
6	5	Output 6.
7	3	Output 7.
8	1	Output 8.

Sample response to `TOUT` (onboard outputs status) command:
```
*TOUT0000_0000
```
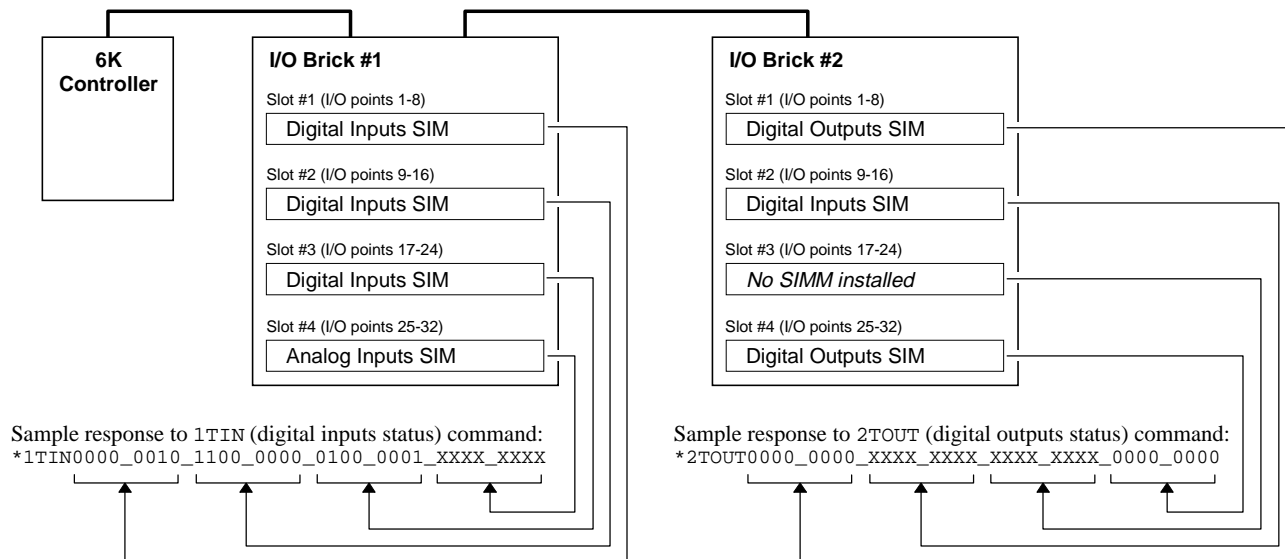
Expansion I/O Bricks

The 6K product allows you to expand your system I/O by connecting up to 8 I/O bricks (see *Installation Guide* for connections). Expansion I/O bricks may be ordered separately (referred to as the "EVM32"). Each I/O brick can hold from 1 to 4 of these I/O SIM modules in any combination:

SIM Type	Programming	Status Report, Assignment
Digital Inputs SIM (8 inputs)	`INFNC, INLVL, INEN, ONIN, INPLC, INSTW`	`TIN, IN`
Digital Outputs SIM (8 outputs)	`OUT, OUTFNC, OUTLVL, OUTEN, OUTALL, OUTPLC, OUTTW, POUT`	`TOUT, [OUT]`
Analog Inputs SIM (8 inputs)	• Enable/Disable: `ANIEN`. • Joystick setup: `JOYAXH, JOYAXL, JOYCDB, JOYCTR, JOYEDB, JOYZ`. • Servo feedback: `ANIFB, SFB` • Following master source: `ANIMAS, FOLMAS`	• Voltage: `TANI, ANI` • Servo position: `TPANI, PANI, FB, TFB`

Each I/O brick has a unique "brick address", denoted with the "`<B>`" symbol in the command syntax. The I/O bricks are connected in series to the "EXPANSION I/O" connector on the 6K. The 1st I/O brick has address #1, the next brick has address #2, and so on. (**NOTE**: If you leave out the brick address in the command, the 6K product will assume you are addressing the command to the onboard I/O.) Each I/O brick has 32 I/O addresses, referenced as <u>absolute</u> I/O point locations:

- SIM slot 1 = I/O points 1-8
- SIM slot 2 = I/O points 9-16
- SIM slot 3 = I/O points 17-24
- SIM slot 4 = I/O points 25-32

Example:

Sample response to `1TIN` (digital inputs status) command:
`*1TIN0000_0010_1100_0000_0100_0001_XXXX_XXXX`

Sample response to `2TOUT` (digital outputs status) command:
`*2TOUT0000_0000_XXXX_XXXX_XXXX_XXXX_0000_0000`

The `TIO` command identifies the connected I/O bricks (and installed SIMs), including the status of each I/O point:

```
*BRICK 1:  SIM Type         Status      Function
     1-8: DIGITAL INPUTS   0000_0000    AAAA_AAAA
    9-16: DIGITAL INPUTS   0000_0000    AAAA_AAAA
   17-24: DIGITAL INPUTS   0000_0000    AAAA_AAAA
   25-32: ANALOG INPUTS     0.000,0.000,0.000,0.000,0.000,0.000,0.000,0.000

*BRICK 2:  SIM Type         Status      Function
     1-8: DIGITAL OUTPUTS  0000_0000    AAAA_AAAA  -- SINKING
    9-16: DIGITAL INPUTS   0000_0000    AAAA_AAAA
   17-24: NO SIM PRESENT
   25-32: DIGITAL OUTPUTS  0000_0000    AAAA_AAAA  -- SOURCING
```

Programming Error Messages

Depending on the error level setting (set with the `ERRLVL` command), when a programming error is created, the 6K controller will respond with an error message and/or an error prompt. A list of all possible error messages is provided in a table below. The default error prompt is a question mark (`?`), but you can change it with the `ERRBAD` command if you wish.

At error level 4 (`ERRLVL4`—the factory default setting) the 6K controller responds with both the error message and the error prompt. At error level 3 (`ERRLVL3`), the 6K controller responds with only the error prompt.

Error Response	Possible Cause
`ACCESS DENIED`	Program security feature enabled, but the *program access input* (`INFNCi-Q` or `LIMFNCi-Q`) is not activated.
`ALREADY DEFINED FOR THUMBWHEELS`	Attempting to assign an I/O function to an I/O that is already defined as a thumbwheel I/O.
`ALTERNATIVE TASK NOT ALLOWED`	Attempting to execute a `LOCK` command directed to another task.
`AXES NOT READY`	Compiled Profile path compilation error.
`COMMAND NOT IMPLEMENTED`	Command is not applicable to the 6K Series product.
`COMMAND NOT ALLOWED IN PROGRAM`	Command is not allowed inside a program definition (between `DEF` and `END`).
`COMMAND/DRIVE MISMATCH`	The command (or ≥ one field in the command) is not appropriate to the `AXSDEF` configuration (e.g., attempting to execute a servo tuning command on a stepper axis)
`ERROR: MOTION ENDS IN NON-ZERO VELOCITY - AXIS N`	Compiled Motion: The last `GOBUF` segment within a `PLOOP`/`PLN` loop does not end at zero velocity, or there is no final `GOBUF` segment placed outside the loop.
`EXCESSIVE PATH RADIUS DIFFERENCE`	Contouring path compilation error.
`FOLMAS NOT SPECIFIED`	No `FOLMAS` for the axis is currently specified. It will occur if `FMCNEW`, `FSHFC`, or `FSHFD` commands are executed and no `FOLMASØ` command was executed, or `FOLMAS0` was executed.
`INCORRECT AXIS`	Axis specified is incorrect.
`INCORRECT BRICK NUMBER`	Attempted to execute a command that addresses an I/O brick that is not connected to your 6K controller.
`INCORRECT DATA`	Incorrect command syntax. Following: Velocity (`V`), acceleration (`A`) or deceleration (`AD`) command is zero (used by `FSHFC` & `FSHFD`).
`INPUT(S) NOT DEFINED AS JOYSTICK INPUT`	Attempted to execute `JOYCDB`, `JOYCTR`, `JOYEDB`, or `JOYZ` before executing `JOYAXH` or `JOYAXL` to assign the analog input to an axis.
`INSUFFICIENT MEMORY`	Not enough memory for the user program or compiled profile segments. This may be remedied by reallocating memory (see `MEMORY` command description).
`INVALID COMMAND`	Command is invalid because of existing conditions

Programming Error Messages *(continued)*

Error Response	Possible Cause

INVALID CONDITIONS FOR COMMAND	System not ready for command (e.g., `LN` command issued before the `L` command).
	
	Following (these conditions can cause an error during Following):
	
	• The `FOLMD` value is too small to achieve the preset distance and still remain within the `FOLRN`/`FOLRD` ratio.
	
	• A phase shift cannot be performed:
	   `FSHFD`.... Error if already shifting or performing other time based move.
	   `FSHFC`.... Error if currently executing a `FSHFD` move, or if currently executing another `FSHFC` move in the opposite direction.
	
	• The `FOLEN1` command was given while a profile was suspended by a `GOWHEN`.

INVALID CONDITIONS FOR S_CURVE ACCELERATION—FIELD *n*	Average (AA) acceleration or deceleration command (e.g., `AA`, `ADA`, `HOMAA`, `HOMADA`, etc.) with a range that violates the equation ½A ≤ AA ≤ A (A is the max. accel or decel command—e.g., `A`, `AD`, `HOMA`, `HOMAD`, etc.)

INVALID DATA	Data for a command is out of range.
	
	Following (these conditions can cause an error during Following):
	
	• The parameter supplied with the command is valid.
	  `FFILT`.... Error if: smooth number is not 0-4
	  `FMCLEN`.. Error if: master steps > 999999999 or negative
	  `FMCP` ...... Error if: master steps > 999999999 or < -999999999
	  `FOLMD`.... Error if: master steps > 999999999 or negative
	  `FOLRD`.... Error if: master steps > 999999999 or negative
	  `FOLRN`.... Error if: follower steps > 999999999 or negative
	  `FSHFC`.... Error if: number is not 0-3
	  `FSHFD`.... Error if: follower steps > 999999999 or < -999999999
	  `GOWHEN`.. Error if: position > 999999999 or < -999999999
	  `WAIT` ...... Error if: position > 999999999 or < -999999999
	
	• Error if a `GO` command is given in the preset positioning mode (`MCØ`) and:
	  `FOLRN` = zero
	  `FOLMD` = zero, or too small
	     (see Following chapter in the *Programmer's Guide*)

INVALID FOLMAS SPECIFIED	Following: An illegal master was specified in `FOLMAS`. A follower may never use its own commanded position or feedback source as its master.
INVALID RATIO	Following: Error if the `FOLRN`:`FOLRD` ratio after scaling is > 127 when a `GO` is executed
INVALID TASK IDENTIFIER	Attempting to launch a `PEXE` or `EXE` command into the supervisor task (task 0).
LABEL ALREADY DEFINED	Defining a program or label with an existing program name or label name
MAXIMUM COMMAND LENGTH EXCEEDED	Command exceeds the maximum number of characters
MAXIMUM COUNTS PER SECOND EXCEEDED	Velocity value is greater than 1,600,000 counts/sec
MOTION IN PROGRESS	Attempting to execute a command not allowed during motion (see Restricted Commands During Motion section in the *Programmer's Guide.*) Following: The `FOLEN1` command was given while that follower was moving in a non-Following mode.

Programming Error Messages *(continued)*

Error Response	Possible Cause
NEST LEVEL TOO DEEP	IFs, REPEATs, WHILEs, or GOSUBs nested greater than 16 levels (for each type)
NO MOTION IN PROGRESS	Attempting to execute a command that requires motion, but motion is not in progress
NO PATH SEGMENTS DEFINED	Compiled Profile compilation error
NO PROGRAM BEING DEFINED	END command issued before a DEF command
NOT ALLOWED IF SFB0	Changes to tuning commands (SGILIM, SGAF, SGI, SGP, SGV, and SGVF) and SMPER are not allowed if SFB0 is selected
NOT ALLOWED IN PATH	Compiled Profile path compilation error
NOT DEFINING A PATH	Executing a compiled profile or contouring path command while not in a path
NOT VALID DURING FOLLOWING MOTION	A GO command was given while moving in the Following mode (FOLEN1) and while in the preset positioning mode (MC0).
NOT VALID DURING RAMP	A GO command was given while moving in a Following ramp and while in the continuous positioning mode (MC1). Following status (FS) bit #3 will be set to 1. A FOLEN command was given during one of these conditions: • During a shift (FSHFC or FSHFD) • During a change in ratio (FOLRN/FOLRD) • During deceleration to a stop
PATH ALREADY MOVING	Compiled Profile path compilation error
PATH NOT COMPILED	Attempting to execute a individual axis profile or a multiple axis contouring path that has not been compiled
PATH RADIUS TOO SMALL	Contouring path compilation error
PATH RADIUS ZERO	Contouring path compilation error
PATH VELOCITY ZERO	Contouring path compilation error
STRING ALREADY DEFINED	A string (program name or label) with the specified name already exists
STRING IS A COMMAND	Defining a program or label that is a command or a variant of a command
UNDEFINED LABEL	Command issued to product is not a command or program name
WARNING: POINTER HAS WRAPPED AROUND TO DATA POINT 1	During the process of writing data (DATTCH) or recalling data (DAT), the pointer reached the last data element in the program and automatically wrapped around to the first datum in the program
WARNING: ENABLE INPUT INACTIVE	**ENABLE** input is no longer connected to ground (**GND**)
WARNING: DEFINED WITH ANOTHER TW/PLC	Duplicate I/O in multiple thumbwheel definitions

Identifying Bad Commands

To facilitate program debugging, the Transfer Command Error (TCMDER) command allows you to transfer the first command that the controller detects as an error. This is especially useful if you receive an error message when running or downloading a program, because it catches and remembers the command that caused the error.

Using Motion Planner:

If you are typing the command in a live terminal emulator session, the controller will detect the bad command and respond with an error message, followed by the ERRBAD error prompt (?). If the bad command was detected on download, the bad command is reported automatically (see example below).

NOTE: If you are not using Motion Planner, you'll have to type in the TCMDER command at the error prompt to display the bad command.

Once a command error has occurred, the command and its fields are stored and system status bit #11 (reported in the TSSF, TSS and SS commands) is set to 1. The status bit remains set until the TCMDER command is issued.

Example Error Scenario:

1. In Motion Planner's program editor, create and save a program with a programming error:

```
DEL badprg   ; Delete a program before defining and downloading
DEF badprg   ; Begin definition of program called badprg
MA11         ; Select the absolute preset positioning mode
A25,40       ; Set acceleration
AD11,26      ; Set deceleration
V5,8         ; Set velocity
VAR1=0       ; Set variable #1 equal to zero
GO11         ; Initiate move on both axes
IF(VAR1<)16  ; MISTYPED IF STATEMENT - should be typed as "IF(VAR1<16)"
VAR1=VAR1+1  ; If variable #1 is less than 16, increment the counter by 1
NIF          ; End IF statement
END          ; End programming of program called badprg
```

2. Using Motion Planner's terminal emulator, download the program to the 6K Series product. Notice that an error response identifies the bad command as an "INCORRECT DATA" item and displays it:

```
> *NO ERRORS
*INCORRECT DATA
> *IF(VAR1<)16
>
```

S-Curve Acceleration/Deceleration Profiling

6K controllers allow you to perform *S-curve* move profiles, in addition to the usual trapezoidal profiles. S-curve profiling provides smoother motion control by reducing the *jerk* (rate of change) in acceleration and deceleration portions of the move profile (see drawing below). Because S-curve profiling reduces jerk, it improves position tracking performance, especially in linear interpolation applications (not contouring).
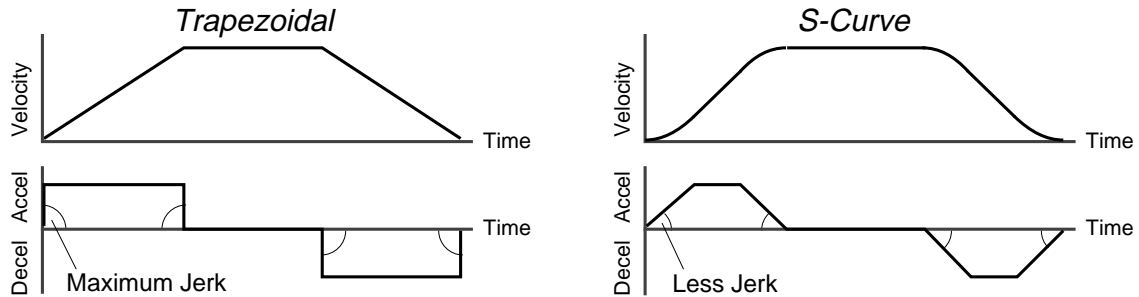
S-Curve Programming Requirements

To program an S-curve profile, you must use the *average accel/decel* commands provided in the 6K Series programming language. For every maximum accel/decel command (e.g., A, AD, HOMA, HOMAD, JOGA, JOGAD, etc.) there is an *average* command for S-curve profiling (see table below).

Maximum Accel/Decel Commands:		Average ("S-Curve") Accel/Decel Commands:	
Command	**Function**	**Command**	**Function**
A	Acceleration	AA	Average Acceleration
AD	Deceleration	ADA	Average Deceleration
HOMA	Home Acceleration	HOMAA	Average Home Acceleration
HOMAD	Home Deceleration	HOMADA	Average Home Deceleration
JOGA	Jog Acceleration	JOGAA	Average Jog Acceleration
JOGAD	Jog Deceleration	JOGADA	Average Jog Deceleration
JOYA	Joystick Acceleration	JOYAA	Average Joystick Acceleration
JOYAD	Joystick Deceleration	JOYADA	Average Joystick Deceleration
LHAD	Hard Limit Deceleration	LHADA	Average Hard Limit Deceleration
LSAD	Soft Limit Deceleration	LSADA	Average Soft Limit Deceleration
PA	Path Acceleration	PAA	Average Path Acceleration
PAD	Path Deceleration	PADA	Average Path Deceleration

Determining the S-Curve Characteristics

The command values for average accel/decel (`AA`, `ADA`, etc.) and maximum accel/decel (`A`, `AD`, etc.) determine the characteristics of the S-curve. To smooth the accel/decel ramps, you must enter average accel/decel command values that satisfy the equation $\boxed{\tfrac{1}{2}\,A \;\leq\; AA \;<\; A,}$ where A represents maximum accel/decel and AA represents average accel/decel. Given this requirement, the following conditions are possible:

Acceleration Setting	Profiling Condition
AA > ½ A, but AA < A	S-curve profile with a variable period of constant acceleration. Increasing the AA value above the pure S-curve level (AA > ½ A), the time required to reach the target velocity and the target distance is decreased. However, increasing AA also increases jerk.
AA = ½ A	*Pure S-curve* (no period of constant acceleration—smoothest motion).
AA = A	Trapezoidal profile (but can be changed to an S-curve by specifying a new AA value less than A).
AA < ½ A; or AA > A	When you issue the `GO` command, the move will not be executed and an error message, `*INVALID CONDITIONS FOR S_CURVE ACCELERATION—FIELD n`, will be displayed.
AA = zero	S-curve profiling is disabled. Trapezoidal profiling is enabled. AA tracks A. (*Track* means the command's value will match the other command's value and will continue to match whatever the other command's value is set to.)
AA ≠ zero and AA ≠ A	S-curve profiling is enabled **only for standard moves** (e.g., not for contouring, which requires the `PADA` and/or `PAA` commands). All subsequent standard moves for that axis must comply with this equation: ½ A ≤ AA < A.
AA > ½ A	Average accel/decel is raised above the pure S-curve level; this decreases the time required to reach the target velocity and distance. However, increasing AA also increases jerk. After increasing AA, you can reduce jerk by increasing A, but be aware that increasing A requires a greater torque to achieve the commanded velocity at the mid-point of the acceleration profile.
No AA value ever entered	Profile will default to trapezoidal. AA tracks A.

If you never change the A or AA <u>deceleration</u> commands, AA deceleration will track AA acceleration. However, once you change A deceleration, AA deceleration will no longer track changes in AA acceleration.

The calculation for determining S-curve average accel and decel move times is as follows (*calculation method identical for S-curve and trapezoidal moves*):

$$\text{Time} = \frac{\text{Velocity}}{A_{avg}} \quad \text{or} \quad \text{Time} = \sqrt{\frac{2 * \text{Distance}}{A_{avg}}}$$

Scaling affects the AA average acceleration (`AA`, `ADA`, etc.) the same as it does for the A maximum acceleration (`A`, `AD`, etc.). See page 16 for details on scaling.

NOTE: Equations for calculating jerk are provided on page 15.

Programming Example (see move profile drawings below)

```
; In this example, axis 1 executes a pure S-curve and takes 1 second
; to reach a velocity of 5 rps; axis 2 executes a trapezoidal profile
; and takes 0.5 seconds to reach a velocity of 5 rps.
SCALE0      ; Disable scaling
DEF SCURV   ; Begin definition of program SCURV
@MA0        ; Select incremental positioning mode
@D40000     ; Set distances to 40,000 positive-
            ; direction steps
A10,10      ; Set max. accel to 10 rev/sec/sec
            ; on axes 1 and 2
AA5,10      ; Set avg. accel to 5 rev/sec/sec on
            ; axis 1, & 10 rev/sec/sec on axis 2
AD10,10     ; Set max. decel to 10 rev/sec/sec
            ; on axes 1 and 2
ADA5,10     ; Set avg. decel to 5 rev/sec/sec on
            ; axis 1, & 10 rev/sec/sec on axis 2
V5,5        ; Set velocity to 5 rps on axes 1 & 2
GO11        ; Execute motion on axes 1 and 2
END         ; End definition of program
```
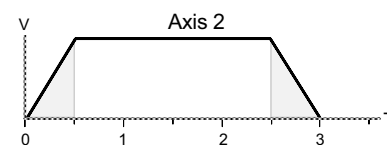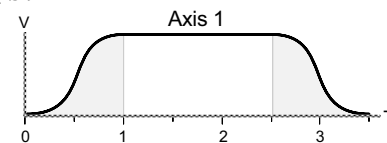

Move profiles

Calculating Jerk
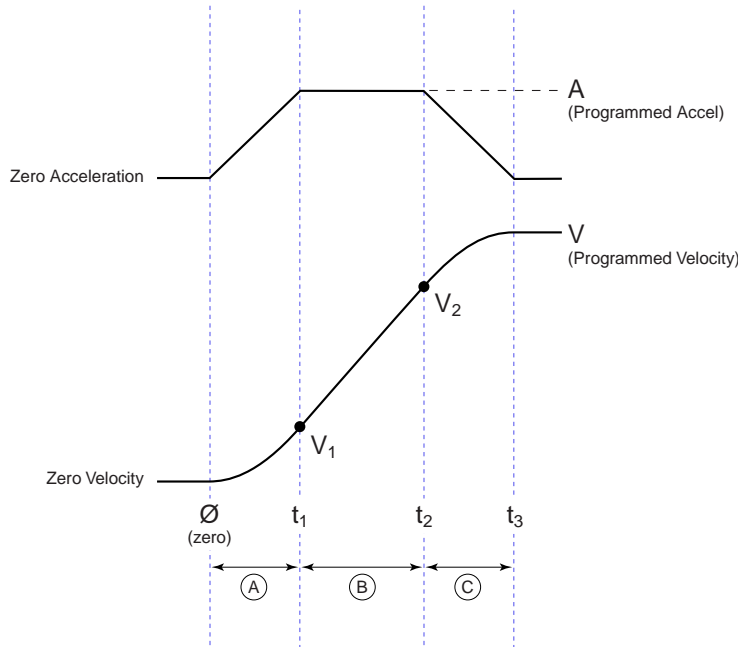
Rules of Motion:

$$\text{Jerk} = \frac{da}{dt}$$

$$a = \frac{dv}{dt}$$

$$v = \frac{dx}{dt} \quad (x = \text{distance})$$

Assuming the accel profile starts when the load is at zero velocity and the ramp to the programmed velocity is not compromised:

$$\textbf{Jerk} = J_A = \frac{A^2 * AA}{V(A-AA)}$$

A = programmed acceleration
    (`A`, `AD`, `HOMAD`, etc.)
AA = average acceleration
    (`AA`, `ADA`, `HOMAA`, etc.)
V = programmed velocity
    (`V`, `HOMV`, etc.)

$$t_1 = \frac{A}{J_A}$$

$$t_2 = \frac{V}{AA} - \frac{A}{J_A}$$

$$t_3 = \frac{V}{AA}$$

NOTE: $t_3 - t_2 = t_1$

$$V_1 = \frac{J_A * t_1^2}{2} = \frac{A^2}{2 * J_A}$$

$$V_2 = V - \frac{A^2}{2 * J_A}$$

(A) $\quad t_1 \geq t \geq \varnothing \quad$ $a(t) = J_A * t$

$$v(t) = \frac{J_A * t^2}{2}$$

$$d(t) = \frac{J_A * t^3}{6}$$

a(t) = acceleration at time t
v(t) = velocity at time t
d(t) = distance at time t

(B) $\quad t_2 \geq t > t_1 \quad$ $a(t) = A$

$$v(t) = \frac{A^2}{2J_A} + A * (t - t_1)$$

$$d(t) = \frac{J_A * t_1^3}{6} + \left(\frac{A * (t - t_1)^2}{2}\right) + \left(V_1 * (t - t_1)\right)$$

(C) $\quad t_3 \geq t > t_2 \quad$ $a(t) = A - (J_A * (t - t_2))$

$$v(t) = V - \left(\frac{J_A * (t_3 - t)^2}{2}\right)$$

$$d(t) = \frac{V^2}{2AA} + \left(\frac{J_A (t_3 - t)^3}{6}\right) - \left(V * (t_3 - t)\right)$$

Starting at a Non-Zero Velocity: If starting the acceleration profile with a non-zero initial velocity, the move comprises two components: a constant velocity component, and an s-curve component. Typically, the change of velocity should be used in the S-curve calculations. Thus, in the calculations above, you would substitute "$(V_F - V_O)$" for "V" (V_F = final velocity, V_O = initial velocity). For example, the jerk equation would be:

$$\textbf{Jerk} = J_A = \frac{A^2 * AA}{(V_F - V_O)(A-AA)}$$

Units of Measure and Scaling

Units of Measure without Scaling

Scaling is disabled (SCALEØ) as the factory default condition:

- Stepper axes: All distance values entered are in commanded counts (sometimes referred to as *motor steps*), and all acceleration, deceleration and velocity values entered are internally multiplied by the DRES command value.

- Servo axes:

Motion Attribute	Units of Measure (per feedback source)	
	Encoder	**Analog Input**
Accel/Decel	Revs/sec/sec *	volts/sec/sec
Velocity	Revs/sec *	volts/sec
Distance	Counts **	Counts **

* All accel/decel & velocity values are internally multiplied by the ERES command value.
** Distance is measured in the counts received from the feedback device.

Contouring & Linear Interpolated Motion: Path acceleration, velocity, and distance are based on the resolution (DRES for steppers, ERES for servos) of axis 1. If multi-tasking is used, path motion units are based on the resolution of the first (lowest number) axis associated with the task (TSKAX).

What is Scaling?

Scaling allows you to program acceleration, deceleration, velocity, and position values in units of measure that are appropriate for your application. The SCALE command is used to enable or disable scaling (SCALE1 to enable, SCALEØ to disable). The motion type(s) you are using in your application determines which scale factor commands you need to configure:

Type of Motion	Accel/Decel Scaling	Velocity Scaling	Distance Scaling
Standard Point-to-Point Motion	SCLA	SCLV	SCLD
Contouring, Linear Interpolation	SCLD	SCLD	SCLD
Following	SCLA	SCLV	SCLD for follower distances SCLMAS for master distances

When Should I Define Scaling Factors?

Scaling calculations are performed when a program is defined or downloaded. Consequently, you must enable scaling (SCALE1) and define the scaling factors (SCLD, SCLA, SCLV, SCLMAS) *prior* to defining (DEF), uploading (TPROG), or running (RUN) the program.

RECOMMENDATION: Place the scaling commands at the beginning of your program file, *before* the location of any defined programs. This ensures that the motion parameters in subsequent programs in your program file are scaled correctly. When you use Motion Planner's Setup Generator wizard, the scaling commands are automatically placed in the appropriate location in your program file.

ALTERNATIVE: Scaling factors could be defined via a terminal emulator *just before* defining or downloading a program. Because scaling command values are saved in battery-backed RAM (remembered until you issue a RESET command), all subsequent program definitions and downloads will be scaled correctly.

NOTES

- Scaling commands are not allowed in a program. If there are scaling commands in a program, the controller will report an error message ("COMMAND NOT ALLOWED IN PROGRAM") when the program is downloaded.

- If you intend to upload a program with scaled motion parameters, be sure to use Motion Planner. Motion Planner automatically uploads the scaling parameters and places them at the beginning of the program file containing the uploaded program from the controller. This ensures correct scaling when the program file is later downloaded.

Acceleration & Deceleration Scaling (`SCLA`*)*

Stepper Axes: If scaling is enabled (`SCALE1`), all accel/decel values entered are internally multiplied by the acceleration scaling factor to convert user units/sec/sec to commanded counts/sec/sec. The scaled values are always in reference in commanded counts, regardless of the existence of an encoder.

Servo Axes: If scaling is enabled (`SCALE1`), all accel/decel values entered are internally multiplied by the acceleration scaling factor to convert user units/sec/sec to encoder or analog input counts/sec/sec.

All accel/decel commands (e.g., A, AA, AD, HOMA, HOMAD, JOGA, etc.) are multiplied by the `SCLA` command value. **NOTE**: Path accel/decel commands (PA, PAD, etc.) are multiplied by the `SCLD` value.

As the accel/decel scaling factor (`SCLA`) changes, the resolution of the accel and decel values and the number of positions to the right of the decimal point also change (see table at right). An accel/decel value with greater resolution than allowed will be truncated (e.g., if scaling is set to `SCLA1Ø`, the `A9.9999` command would be truncated to `A9.9`).

`SCLA` value **(counts/unit/unit)**	**Decimal Places**
1 - 9 0	
10 - 99 1	
100 - 999 2	
1000 - 9999 3	
10000 - 99999............................ 4	
100000 - 999999........................ 5	

The following equations can help you determine the range of acceleration and deceleration values.

Axis Type	Min. Accel or Decel (resolution)	Max. Accel or Decel
Stepper	$\dfrac{0.001 * \text{DRES}}{\text{SCLA}}$	$\dfrac{999.9999 * \text{DRES}}{\text{SCLA}}$
Servo	Encoder Feedback: $\dfrac{0.001 * \text{ERES}}{\text{SCLA}}$	Encoder Feedback: $\dfrac{999.9999 * \text{ERES}}{\text{SCLA}}$
	ANI Feedback: * $\dfrac{0.205}{\text{SCLA}}$	ANI Feedback: * $\dfrac{204799.9795}{\text{SCLA}}$

* This calculation assumes the analog input range (`ANIRNG` value) is left in its default setting (range is -10V to +10V).

Velocity Scaling (SCLV)

Stepper Axes: If scaling is enabled (SCALE1), all velocity values entered are internally multiplied by the velocity scaling factor to convert user units/sec to commanded counts/sec. The scaled values are always in reference to commanded counts (sometimes referred to as "motor steps").

Servo Axes: If scaling is enabled (SCALE1), all velocity values entered are internally multiplied by the velocity scaling factor to convert user units/sec to encoder or analog input counts/sec.

All velocity commands (e.g., V, HOMV, HOMVF, JOGVH, JOGVL, etc.) are multiplied by the SCLV command value. **NOTE**: Path velocity (PV) is multiplied by the SCLD value.

As the velocity scaling factor (SCLV) changes, the velocity command's range and its decimal places also change (see table below). A velocity value with greater resolution than allowed will be truncated. For example, if scaling is set to SCLV10, the V9.9999 command would be truncated to V9.9.

SCLV Value (counts/unit)	Velocity Resolution (units/sec)	Decimal Places
1 - 9	1	0
10 - 99	0.1	1
100 - 999	0.01	2
1000 - 9999	0.001	3
10000 - 99999	0.0001	4
100000 - 999999	0.00001	5

Use the following equations to determine the maximum velocity range for your product type.

Max. Velocity for Stepper Axes	Max. Velocity for Servo Axes (determined by feedback source selected for axis #1)
$\dfrac{n}{SCLV}$ n = maximum velocity is determined by the PULSE command setting.	Encoder Feedback: $\dfrac{6{,}500{,}000}{SCLV}$ ANI Feedback: * $\dfrac{1000 * 205}{SCLV}$

* This calculation assumes the analog input range (ANIRNG value) is left in its default setting (range is -10V to +10V).

Distance Scaling (SCLD and SCLMAS)

Stepper Axes: If scaling is enabled (SCALE1), all distance values entered are internally multiplied by the distance scaling factor to convert user units to commanded counts ("motor steps").

Servo Axes: If scaling is enabled (SCALE1), all distance values entered are internally multiplied by the distance scaling factor to convert user units to encoder or analog input counts.

All distance commands (e.g., D, PSET, REG, SMPER) are multiplied by the SCLD command value. The only exception is for master distance values (see table below)

Scaling for Following Motion: The SCLD command defines the follower axis distance scale factor, and the SCLMAS command defines the master's distance scale factor. The Following-related commands that are affected by SCLD and SCLMAS are listed in the table below.

Commands Affected by Master Scaling (SCLMAS)	Commands Affected by Follower Scaling (SCLD)
FMCLEN: *Master Cycle Length*	FOLRN: *Follower-to-Master Ratio (Numerator)*
FMCP: *Master Cycle Position Offset*	FGADV: *Geared Advance*
FOLMD: *Master Distance*	FSHFD: *Preset Phase Shift*
FOLRD: *Follower-to-Master Ratio (Denominator)*	GOWHEN: *Conditional GO* (left-hand variable ≠ PMAS)
GOWHEN: *Conditional GO* (left-hand variable is PMAS)	TPSHF & [PSHF]: *Net Position Shift of Follower*
TPMAS & [PMAS]: *Position of Master Axis*	TPSLV & [PSLV]: *Position of Follower Axis*
TVMAS & [VMAS]: *Velocity of Master Axis*	

As the `SCLD` or `SCLMAS` scaling factor changes, the distance command's range and its decimal places also change (see table below). A distance value with greater resolution than allowed will be truncated. For example, if scaling is set to `SCLD4000`, the `D105.2776` command would be truncated to `D105.277`.

SCLD or SCLMAS **Value** (counts/unit)	**Distance Resolution** (units)	**Distance Range *** (units)	**Decimal Places**
1 - 9	1.0	0 - ±999999999	0
10 - 99	0.10	0.0 - ±99999999.9	1
100 - 999	0.010	0.00 - ±9999999.99	2
1000 - 9999	0.0010	0.000 - ±999999.999	3
10000 - 99999	0.00010	0.0000 - ±99999.9999	4
100000 - 999999	0.00001	0.00000 - ±9999.99999	5

NOTE   FRACTIONAL STEP TRUNCATION   NOTE

If you are operating in the incremental mode (`MAØ`), or specifying master distance values with `FOLMD`, when the distance scaling factor (`SCLD` or `SCLMAS`) and the distance value are multiplied, a fraction of one step may be left over. This fraction is truncated when the distance value is used in the move algorithm. This truncation error can accumulate over a period of time, when performing incremental moves continuously in the same direction. To eliminate this truncation problem, set `SCLD` or `SCLMAS` to 1, or a multiple of 10.

Scaling Example — Stepper Axes

Axis #1 and axis #2 control 25,000 step/rev motor/drives attached to 5-pitch leadscrews. The user wants to program motion parameters in inches; therefore the scale factor calculation is: 25,000 steps/rev x 5 revs/inch = 125,000 steps/inch. For instance, with a scale factor of 125,000, the operator could enter a move distance value of 2.000 and the controller would send out 250,000 pulses, corresponding to two inches of travel.

```
SCALE1                 ; Enable scaling
DRES25000,25000        ; Set drive resolution to 25,000 steps/rev on both axes
SCLD125000,125000      ; Allow user to enter distance in inches (both axes)
SCLV125000,125000      ; Allow user to enter velocity in inches/sec (both axes)
SCLA125000,125000      ; Allow entering accel/decel in inches/sec/sec (both axes)
```

Scaling Example — Servo Axes

Axis #1 controls a 4,000 count/rev servo motor/drive system (using a 1000-line encoder) attached to a 5-pitch leadscrew. The user wants to position in inches; therefore, the scale factor calculation is 4,000 counts/rev x 5 revs/inch = 20,000 counts/inch. Half way through the motion process, axis #1 must switch to ANI feedback for the purpose of positioning to a voltage (scale factor is 205 counts/volt).

Axis #2 controls a 4,000 count/rev servo motor/drive system (using a 1000-line encoder) attached to a 10-pitch leadscrew. The user wants to position in inches (scale factor calculation: 4,000 counts/rev x 10 revs/inch = 40,000 counts/inch).

```
SFB1,1          ; Select encoder feedback for both axes
ERES4000,4000   ; Set encoder res to 4000 steps/rev (post quadrature)
SCALE1          ; Enable scaling
SCLD20000,40000 ; Allow user to enter distance values in inches
SCLV20000,40000 ; Allow user to enter velocity values in inches/sec
SCLA20000,40000 ; Allow user to enter accel/decel values in inches/sec/sec
SFB2            ; Select ANI feedback for axis #1
SCALE1          ; Enable scaling
SCLD205         ; Allow user to enter distance values in volts
SCLV205         ; Allow user to enter velocity values in volts/sec
SCLA205         ; Allow user to enter accel/decel values in volts/sec/sec
SFB1,1          ; Select encoder feedback for both axes (prepare for motion)
```

Scaling Example — Following

Typically, the master and follower scale factors are programmed so that master and follower units are the same, but this is not required. Consider the scenario below as an example.

The master is a 1000-line encoder (4000 counts/rev post-quadrature) mounted to a 50 teeth/rev pulley attached to a 10 teeth/inch conveyor belt, resulting in 80 counts/tooth (4000 counts/50 teeth = 80 counts/tooth). To program in inches, you would set up the master scaling factor with the `SCLMAS800` command (80 counts/tooth ∗ 10 teeth/inch = 800 counts/inch).

The follower axis is a servo motor with position feedback from a 1000-line encoder (4000 counts/rev). The motor is mounted to a 4-pitch (4 revs/inch) leadscrew. Thus, to program in inches, you would set up the follower scaling factor with the `SCLD16000` command (4000 counts/rev ∗ 4 revs/inch = 16000 counts/inch).

```
SCALE1      ; Enable scaling
SCLMAS800   ; Master scaling (80 counts/tooth * 10 teeth/inch = 800 counts/inch)
SCLD16000   ; Follower scaling (4000 counts/rev * 4 revs/inch = 16000 counts/inch)
```

Scaling Example — Contouring & Linear Interpolation

This simple example uses 2 servo axes (axes 1 and 2) for contouring. Both axes use encoder feedback with a resolution (`ERES`) of 4000 counts/rev, axis 1 uses a 10-pitch (10 revs per inch) leadscrew and axis 2 uses a 5-pitch (5 revs per inch) lead screw, and you want to program in inches. For this application you would use the `SCLD40000,20000` command to establish path motion units in inches: distance is inches, acceleration is inches/sec/sec, and velocity is inches/sec. Note that all path motion attributes are scaled by the `SCLD` value.

```
SCALE1            ; Enable scaling
SCLD40000,20000   ; Set scaling to program in inches:
                  ; Axis 1: 4000 counts/rev * 10 revs/inch = 40000 counts/inch
                  ; Axis 2: 4000 counts/rev * 5 revs/inch = 20000 counts/inch
PV5               ; Set path velocity to 5 inches/sec
PA50              ; Set path acceleration to 50 inches/sec/sec
PAD100            ; Set path deceleration to 100 inches/sec/sec
DEF prog1         ; Begin definition of path named prog1
PAXES1,2          ; Set axes 1 and 2 as the X and Y contouring axes
PAB0              ; Set to incremental coordinates
PLIN1,1           ; Specify X-Y endpoint position to create a 45 degree
                  ; angle line segment
END               ; End definition of path prog1
PCOMP prog1       ; Compile path prog1
PRUN prog1        ; Execute path prog1
```

www.comoso.com

%   Task Identifier

Type	Multi-Tasking	**Product**	**Rev**
Syntax	i%<command>	6K	5.0
Units	i = task number		
Range	1-10		
Default	1		
Response	n/a		
See Also	LOCK, [SWAP], [TASK], TSKAX, TSKTRN, TSWAP, TTASK		

Use the Task Identifier (%) prefix to specify that the associated command will **affect** the indicated task number. For most simple multi-tasking applications, the % prefix is used to start a program running in a specific task. For example, the drawing on the right illustrates how the 1%move1 command starts the program called "move1" in task 1 (specified with the 1% prefix).

6K Controller
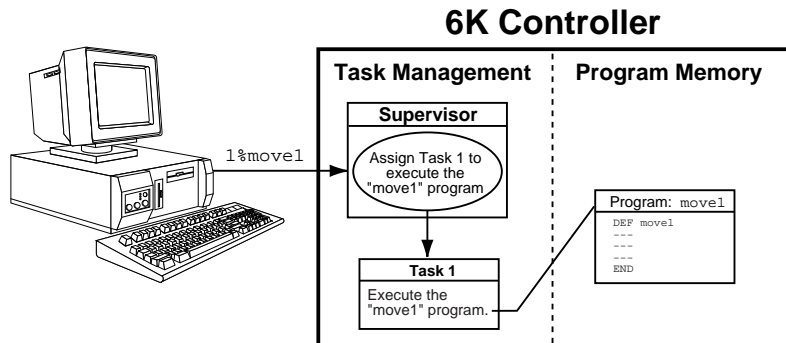
Because the % prefix specifies the task number that the associated command will affect, new tasks can be started from within other tasks, as shown in the drawing on the right.

Within a program in a task, it is not necessary to use the % prefix unless trying to initiate a program or command in a different task. For example, if the fill program running in task 3 executes a COMEXC1 command, only task 3 is placed into COMEXC1 mode. If the fill program running in task 3 also executes a 2%PS command, task 3 executes the command, but the program being executed in task 2 is paused, not task 3.

6K Controller
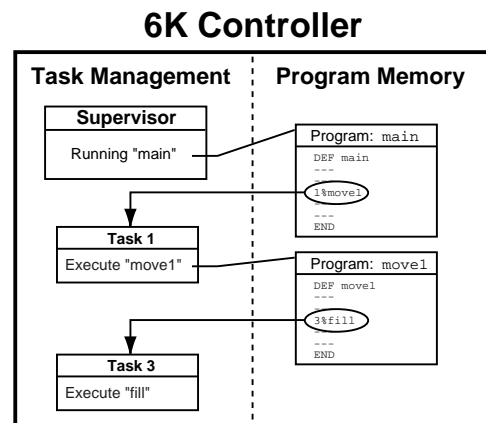
How the Task Supervisor Works: The "Task Supervisor" (also referred to as Task ∅) is the main program execution environment. It contains the command buffer and parser. Immediate commands and commands executed from the communications buffer are implicitly directed to affect the supervisor unless explicitly directed to a task with the % prefix. Only the supervisor executes buffered commands from the communications buffer. If the supervisor is executing a program, incoming commands will be buffered, not executed. If the supervisor is not executing a program, it will execute commands from the input command buffer, even if the other tasks are executing programs. If a command in the command buffer has a task prefix, it is still executed by the supervisor, but affects the task specified by the prefix.

[!]   Immediate Command Identifier

Type	Operator (Other)	**Product**	**Rev**
Syntax	!<command>	6K	5.0
Units	n/a		
Range	n/a		
Default	n/a		
Response	n/a		
See Also	COMEXC		

The Immediate Command Identifier (!) changes a buffered command into an immediate command. All immediate commands are processed immediately, even before previously entered buffered commands.

All 6K Series commands are buffered.

The commands that use the ! identifier are identified in the **Syntax** portion of the command description.

> **NOTE**
> A command with the ! prefix cannot be stored in a program.

[@]   Global Command Identifier

Type	Operator (Other)	**Product**	**Rev**
Syntax	@<command><field1>	6K	5.0
Units	n/a		
Range	n/a		
Default	n/a		
Response	n/a		
See Also	INDAX		

The Global Command Identifier (@) is used to set the value of all fields to the value entered only in the first field. For example, @A1 assigns the value 1 to all axes. All commands with multiple fields are able to use the Global Command Identifier. If you have any doubts about which commands can use the @ symbol, refer to the **Syntax** portion of the command description.

;   Begin Comment

Type	Operator (Other)	**Product**	**Rev**
Syntax	;<this is a comment>	6K	5.0
Units	n/a		
Range	n/a		
Default	n/a		
Response	n/a		
See Also	None		

The Begin Comment (;) command is used to comment application programs. The comment begins with a semicolon (;) and is terminated by a command delimiter. The comment is not stored in a program. An example of using the comment delimiter is as follows:

```
DEF pick ; Begin definition of program pick<cr>
```

$ Label Declaration

Type	Operator (Other)	**Product**	**Rev**
Syntax	`<!>$<t>`	6K	5.0
Units	t = text name		
Range	Text name of 6 characters or less		
Default	n/a		
Response	n/a		
See Also	DEF, DEL, END, GOSUB, GOTO, JUMP, RUN, TLABEL		

The Label Declaration ($) command defines the current location as the label specified. A label consists of 6 or fewer alpha-numeric characters and must start with an alpha-character, not a number. Labels can only be defined within a program or subroutine. The GOTO, GOSUB or JUMP commands can be used to branch to a label. The RUN command can also be used to start executing statements at a label. The label cannot be deleted by a DEL command. However, when the program that contains the label is deleted, all labels contained within the program will be deleted.

NOTE: The maximum number of labels possible is 600.

A label declaration cannot consist of any of the following characters:
!, _, #, $, %, ^, &, *, (,), +, −, {, }, \, |, ", :, ;, ', <, >, ,, ., ?, /, =

NOTE: A label cannot have the same name as a 6K Series command. For example, $A and $A123 are illegal labels.

Example
```
DEF pick        ; Begin definition of program called pick
GO1100          ; Initiate motion on axes 1 and 2
IF(VAR1=5)      ; If variable 1 = 5 then do commands between IF and ELSE,
                ; otherwise commands between ELSE and NIF
GOTO pick1      ; Goto label pick1
ELSE            ; Else part of IF command
GOTO pick2      ; Goto label pick2
NIF             ; End IF command
$pick1          ; Label declaration for pick1
GO0011          ; Initiate motion on axes 3 and 4
BREAK           ; Break out of current subroutine or program
$pick2          ; Label declaration for pick2
GO1001          ; Initiate motion on axes 1 and 4
END             ; End program definition
RUN pick        ; Execute program named pick
```

[#]   Step Through a Program

Type	Operator (Other)		
		Product	**Rev**
Syntax	!#<i>	6K	5.0
Units	i = number of commands to execute from the buffer		
Range	i = 1 – 200		
Default	1		
Response	n/a		
See Also	DEF, HELP, STEP, TRACE, TRANS		

This command controls the execution of a program or sequence when the single step mode is enabled (STEP1). Each time you enter the !#<i> command followed by a delimiter, i commands in the sequence buffer will be executed. A !# followed by a delimiter will cause one command to be executed.

Single step mode can be advantageous when trying to debug a program.

Example:
```
DEF tst        ; Begin definition of program named tst
@V1            ; Set velocity to 1 unit/sec on all axes
@A10           ; Set acceleration to 10 units/sec/sec on all axes
D1,2,3,4       ; Set distance to 1 unit on axis 1, 2 units on axis 2,
               ; 3 units on axis 3, and 4 units on axis 4
GO1101         ; Initiate motion on axes 1, 2, and 4
OUT11X1        ; Turn on on-board programmable outputs 1, 2, and 4,
               ; leave 3 unchanged
END            ; End program definition
STEP1          ; Enable single step mode
RUN tst        ; Execute program named tst
```

NOTE: After entering the command RUN no action will occur because single step mode has been enabled. Single step operation is as follows:

```
!#2            ; First 2 commands in the program tst are executed,
               ; commands to be executed are @V1 and @A10.
!#             ; Execute 1 command from program; command to execute is D1,2,3,4
!#1            ; Execute 1 command from program; command to be executed is GO1101
!#2            ; Execute 2 commands from program; commands to be executed are
               ; OUT11X1 and END
```

'   Enter Interactive Data

Type	Operator (Other)		
		Product	**Rev**
Syntax	!'<numeric data>	6K	5.0
Units	Numeric data is command-dependent		
Range	Numeric data is command-dependent		
Default	n/a		
Response	n/a		
See Also	[READ], VARI, VARS		

To enter data interactively, two operations must occur. First, numeric information must be requested. Requesting the numeric information is accomplished with the VARx=READy command. The x specifies the numeric variable to place the data into, and the y specifies the string variable to transmit before the data is entered. Numeric information can also be requested by placing the READ command in place of a command argument (e.g., A(READ1),12.52,(READ2),5.62). After the data has been requested, a numeric response must be provided. The numeric response must be preceded by the interactive data specifier (!') and followed by a delimiter (<cr> or <lf>).

Command processing will pause while waiting for data.

Example:
```
VARS1="Enter the count > "  ; Set string variable 1 equal to the message
VAR5=READ1     ; Transmit string variable 1, and wait for numeric data in the
               ; form of !'<data>. Once numeric data has been received, place
               ; it in numeric variable 5
!'65.12        ; Variable 5 will receive the value 65.12
```

[.] Bit Select

			Product	Rev
Type	Operator (Other)		6K	5.0
Syntax	<command>.i			
Units	i = bit number			
Range	Command-dependent			
Default	None			
Response	n/a			
See Also	[AS], [ER], ERROR, [IN], INEN, INLVL, [INO], INTHW, LHLVL, [LIM], [MOV], ONIN, ONUS, OUT, OUTEN, OUTLVL, POUT, [SS], TAS, TER, TIN, TINO, TINT, TLIM, TOUT, TSS, TUS, [US]			

The Bit Select (.) operator specifies which bit to select. The primary purpose of this command is to let the user specify a specific bit (or range), instead of having to type in an entire bit string.

When using the bit operator in a comparison, the bit operator must always come to the left of the comparison. For example, the command IF(1AS.12=b1) is legal, but IF(b1=1AS.12) is illegal.

Command Shortcut Examples (affect only one binary bit location)**:**

- Activate outputs at I/O location Brick 3, I/O point 9:   3OUT.9=1
- Enable analog input at I/O location Brick 2, I/O point 2: 2ANIEN.2=E
- Enable error-checking bit 6 for task 3:   3%ERROR.6=1

Example:
```
VARB2=ER.12      ; Error status bit 12 assigned to binary variable 2
VARB2            ; Response (if bit 12 is set to 1):
                 ; *VARB2=XXXX_XXXX_XXX1_XXXX_XXXX_XXXX_XXXX_XXXX
2OUT.5=1         ; Activate the output at location Brick 2, I/O point 5
```

["] Begin and End String

			Product	Rev
Type	Operator (Other)		6K	5.0
Syntax	"<message>" (see below for possibilities)			
Units	n/a			
Range	n/a			
Default	n/a			
Response	n/a			
See Also	DWRITE, VARS, WRITE, WRVARS			

There are three commands that deal with string variables, or messages. The first of these commands is the VARS command. This command sets a string variable equal to a specific message (e.g., VARS1="Enter part count"). The message must be placed in quotes for it to be recognized. The same can be said for the WRITE and DWRITE commands. Their messages must also be placed in quotes (e.g., WRITE"Today is the first day of the rest of your life").

Syntax possibilities: VARSn="<message>" where n equals the string variable number
                        WRITE"<message>"
                        DWRITE"<message>"

There are three ASCII characters that cannot be used within the quotes (:, ", and ;). These characters can be specified in the string by using the backslash character (\) in combination with the ASCII decimal value for the character. For example, if you wanted to display the message "WHY ASK WHY" in quotes, you would use the following syntax: WRITE"\34WHY ASK WHY\34".

An ASCII table is provided in Appendix B. Common characters and their ASCII equivalent value:

Character	Description	ASCII Decimal Value
<lf>	Line Feed	10
<cr>	Carriage Return	13
"	Quote	34
:	Colon	58
;	Semi-colon	59
\	Backslash	92 (cannot be used with DWRITE)

[\ ]   ASCII Character Designator

Type	Operator(Other)		**Product**	**Rev**
Syntax	See below		6K	5.0
Units	n/a			
Range	n/a			
Default	n/a			
Response	n/a			
See Also	VARS, WRITE, WRVARS			

The ASCII Character Designator (\) operator is used to place a character in a string that is normally not represented by a keyboard character. The (\) operator can be used within the VARS or the WRITE commands. The syntax for the (\) operator is as follows:

WRITE"\<i>", Where <i> is the ASCII decimal equivalent of the character to be placed in the string.

VARS1="\<i>", Where <i> is the ASCII decimal equivalent of the character to be placed in the string.

There are three ASCII characters that cannot be used within the quotes (:, ;, and "). These characters must be specified in the string by using the backslash character (\) in combination with the ASCII decimal value for the character.

An ASCII table is provided in Appendix B. Common characters and their ASCII equivalent value:

Character	Description	ASCII Decimal Value
<lf>	Line Feed	10
<cr>	Carriage Return	13
"	Quote	34
:	Colon	58
;	Semi-colon	59
\	Backslash	92

Example:
```
WRITE"cd\92AT6400\13\10"   ;Displays: cd\AT6400<cr><lf>
```

[=]   Assignment or Equivalence

Type	Operator (Mathematical or Relational)		**Product**	**Rev**
Syntax	See below		6K	5.0
Units	n/a			
Range	n/a			
Default	n/a			
Response	n/a			
See Also	[>], [>=], [<], [<=], [<>], [AND], IF, [OR], UNTIL, VAR, VARB, VARI, VARS, WAIT, WHILE			

The assignment or equivalence operator (=) is used to either assign a value to a variable, or compare two values and/or variables. The (=) operator is limited to 1 assignment operation per line. It is acceptable to state VAR1=25, but it is unacceptable to state VAR1=25=VAR2.

More than 1 equivalence operator can be used in a command; however, the total number of relational operators used in a line is limited by the command length limitation (80 characters), not the number of relational operators (e.g., the command IF(VAR1=1 AND VAR2=4 AND VAR3=4) is a legal command).

When (=) is used as an assignment operator, it can be used with these commands: VAR, VARI, VARB, VARS. When (=) is used as an equivalence operator, it can be used with these commands: IF, WHILE, UNTIL, WAIT.

[>]   Greater Than

			Product	Rev
Type	Operator (Relational)		6K	5.0
Syntax	See below			
Units	n/a			
Range	n/a			
Default	n/a			
Response	n/a			
See Also	[=], [>=], [<], [<=], [<>], [AND], IF, [OR], UNTIL, WAIT, WHILE			

The greater than (>) operator is used to compare two values. If the value on the left of the operator is greater than the value on the right of the operator, then the expression is *TRUE*. If the value on the left is less than or equal to the value on the right of the operator, then the expression is *FALSE*. The greater than operator (>) can only be used to compare two values.

More than one (>) operator can be used within a single command; however, the total command length is limited to 80 characters.

The (>) operator can be used in conjunction with the IF, WHILE, UNTIL, and WAIT commands.

Examples of valid commands are IF(VAR1>1) and WHILE(VAR1>1 AND VAR2>3). An example of an invalid command is IF(5>VAR1>1).

[>=]   Greater Than or Equal

			Product	Rev
Type	Operator (Relational)		6K	5.0
Syntax	See below			
Units	n/a			
Range	n/a			
Default	n/a			
Response	n/a			
See Also	[=], [>], [<], [<=], [<>], [AND], IF, [OR], UNTIL, WAIT, WHILE			

The greater than or equal (>=) operator is used to compare two values. If the value on the left of the operator is greater than or equal to the value on the right of the operator, then the expression is *TRUE*. If the value on the left is less than the value on the right of the operator, then the expression is *FALSE*. The greater than or equal operator (>=) can only be used to compare two values.

More than one (>=) operator can be used within a single command; however, the total command length is limited to 80 characters.

The (>=) operator can be used in conjunction with the IF, WHILE, UNTIL, and WAIT commands.

Examples of valid commands are IF(VAR1>=1) and WHILE(VAR1>=1 AND VAR2>=3). An example of an invalid command is IF(5>VAR1>=1).

[<]   Less Than

			Product	Rev
Type	Operator (Relational)		6K	5.0
Syntax	See below			
Units	n/a			
Range	n/a			
Default	n/a			
Response	n/a			
See Also	[=], [>], [>=], [<=], [<>], [AND], IF, [OR], UNTIL, WAIT, WHILE			

The less than (<) operator is used to compare two values. If the value on the left of the operator is less than the value on the right of the operator, then the expression is *TRUE*. If the value on the left is greater than or equal to the value on the right of the operator, then the expression is *FALSE*. The less than operator (<) can only be used to compare two values.

More than one (<) operator can be used within a single command; however, the total command length is limited to 80 characters.

The (<) operator can be used in conjunction with the IF, WHILE, UNTIL, and WAIT commands.

Examples of valid commands are IF(VAR1<1) and WHILE(VAR1<1 AND VAR2<3). An example of an invalid command is IF(1<VAR1<54).

[<=] Less Than or Equal

Type	Operator (Relational)		Product	Rev
Syntax	See below		6K	5.0
Units	n/a			
Range	n/a			
Default	n/a			
Response	n/a			
See Also	[=], [>], [<], [>=], [<>], [AND], IF, [OR], UNTIL, WAIT, WHILE			

The less than or equal (<=) operator is used to compare two values. If the value on the left of the operator is less than or equal to the value on the right of the operator, then the expression is *TRUE*. If the value on the left is greater than the value on the right of the operator, then the expression is *FALSE*. The less than or equal operator (<=) can only be used to compare two values.

More than one (<=) operator can be used within a single command; however, the total command length is limited to 80 characters.

The (<=) operator can be used in conjunction with the IF, WHILE, UNTIL, and WAIT commands.

Examples of valid commands are IF(VAR1<=1) and WHILE(VAR1<=1 AND VAR2<=3). An example of an invalid command is IF(1<VAR1<=54).

[<>] Not Equal

Type	Operator (Relational)		Product	Rev
Syntax	See below		6K	5.0
Units	n/a			
Range	n/a			
Default	n/a			
Response	n/a			
See Also	[=], [>=], [<], [<=], [AND], IF, [OR], UNTIL, WAIT, WHILE			

The not equal (<>) operator is used to compare two values. If the value on the left of the operator is not equal to the value on the right of the operator, then the expression is *TRUE*. If the value on the left is equal to the value on the right of the operator, then the expression is *FALSE*. The not equal operator (<>) can only be used to compare two values.

More than one (<>) operator can be used within a single command; however, the total command length is limited to 80 characters.

The (<>) operator can be used in conjunction with the IF, WHILE, UNTIL, and WAIT commands.

Examples of valid commands are IF(VAR1<>1) and WHILE(VAR1<>1 AND VAR2<=3). An example of an invalid command is IF(1<VAR1<>54).

[( )]   Operation Priority Level

			Product	Rev
Type	Operator (Mathematical)		6K	5.0
Syntax	See below			
Units	n/a			
Range	n/a			
Default	n/a			
Response	n/a			
See Also	[=], [-], [*], [/], [SQRT], VAR, VARI			

The Operation Priority Level operators determines which operation to do first in a mathematical expression. For example, if you want to add 5 to 6 times 3, you can specify `VAR1=6*3+5` or `VAR1=5 + (6*3)`.

More than one set of parentheses can be used in a mathematical expression; however, they cannot be nested (e.g. `VAR1=(VAR2 * 3) * (3 + VAR4)`).

[+]   Addition

			Product	Rev
Type	Operator (Mathematical)		6K	5.0
Syntax	See below			
Units	n/a			
Range	n/a			
Default	n/a			
Response	n/a			
See Also	[=], [( )], [-], [*], [/], [SQRT], VAR, VARI, VARB			

The addition (+) operator adds the value to the left of the operator with the value to the right of the operator. The addition operator can only be used in conjunction with the `VAR`, `VARI` and `VARB` commands.

The total command length must be less than 80 characters. The order of precedence is **left to right**. The Operation Priority Level (`()`) operators can be used; however, they cannot be nested.

Examples of valid commands: `VAR1=1+2+3+4+5+6+7+8+9`
`VAR2=VAR1+1+(5*3)`
`VARB1=b1101 + b11001`

[−]   Subtraction

			Product	Rev
Type	Operator (Mathematical)		6K	5.0
Syntax	See Below			
Units	n/a			
Range	n/a			
Default	n/a			
Response	n/a			
See Also	[=], [( )], [+], [*], [/], [SQRT], VAR, VARI, VARB			

The subtraction (-) operator subtracts the value to the right of the operator from the value to the left of the operator. The subtraction operator can only be used in conjunction with the `VAR`, `VARI` and `VARB` commands.

The total command length must be less than 80 characters. The order of precedence is **left to right**. The Operation Priority Level (`()`) operators can be used; however, they cannot be nested.

Examples of valid command s: `VAR1=1-2-3-4-5-6-7-8-9`
`VAR2=VAR1-1+(5*3)`
`VARB1=b111101 - b11001`

[*]　　　　　Multiplication

Type	Operator (Mathematical)	**Product**	**Rev**
Syntax	See Below	6K	5.0
Units	n/a		
Range	n/a		
Default	n/a		
Response	n/a		
See Also	[=], [( )], [+], [-], [/], [SQRT], VAR, VARI, VARB		

The multiplication (*) operator multiplies the value to the right of the operator with the value to the left of the operator. The multiplication operator can only be used in conjunction with the VAR, VARI and VARB commands (VARI integer values are truncated).

The total command length must be less than 80 characters. The order of precedence is **left to right**. The Operation Priority Level (( )) operators can be used; however, they cannot be nested.

Examples of valid commands:　`VAR1=1*2*3*4*5*6*7*8*9`
　　　　　　　　　　　　　`VAR2=VAR1-1+(5*3)`
　　　　　　　　　　　　　`VARB1=b111101 * b11001`

[/]　　　　　Division

Type	Operator (Mathematical)	**Product**	**Rev**
Syntax	See Below	6K	5.0
Units	n/a		
Range	n/a		
Default	n/a		
Response	n/a		
See Also	[=], [( )], [+], [-], [*], [SQRT], VAR, VARI, VARB		

The division (/) operator divides the value to the left of the operator by the value on the right of the operator. The result of the division is specified to five decimal places (VARI integer variables are truncated). The division operator can only be used in conjunction with the VAR and VARB commands.

The total command length must be less than 80 characters. The order of precedence is **left to right**. The Operation Priority Level (( )) operators can be used; however, they cannot be nested.

Examples of valid commands:　`VAR1=1/2/3/4/5/6/7/8/9`
　　　　　　　　　　　　　`VAR2=VAR1-1/(5*3)`
　　　　　　　　　　　　　`VARB1=b111101 / b11001`　　　**DIVISION BY ZERO IS NOT ALLOWED.**

[&]　　　　　Boolean And

Type	Operator (Bitwise)	**Product**	**Rev**
Syntax	See Below	6K	5.0
Units	n/a		
Range	n/a		
Default	n/a		
Response	n/a		
See Also	[=], [|], [~], [^], [<<],[>>], VAR, VARI, VARB		

The Boolean And (&) operator performs a logical AND on the two values to the left and right of the operator when used with the VAR or VARI command. The Boolean And (&) performs a bitwise AND on the two values to the left and right of the operator when used with the VARB command.

For a logical AND (using VAR or VARI), the possible combinations are as follows:

positive number & positive number	=	1
positive number & zero or a negative number	=	0
zero or negative number & positive number	=	0
zero or negative number & zero or negative number	=	0

　　　Example:　`VAR1=5 & -1`
　　　Result:　`VAR1=0`

For a bitwise AND (using VARB), the value on the left side of the & operator has each of its bits ANDed with the corresponding bit of the value on the right side of the operator. Each bit comparison will be composed of 9 possible combinations:

```
1 & 1 = 1              1 & X = X
1 & 0 = 0              X & 1 = X
0 & 1 = 0              0 & X = 0
0 & 0 = 0              X & 0 = 0
X & X = X
```

Example: VARB1=b0000 1000 & b1000 1011 1
Response to VARB1 is *VARB1=0000_1000_XXXX_XXXX_XXXX_XXXX_XXXX_XXXX

Example: VARB1=h32FD & h23
Response to VARB1 is *VARB1=0100_0100_0000_0000_0000_0000_0000_0000

Example: VARB1=h23 & b1101
Response to VARB1 is *VARB1=0100_XX00_0000_0000_0000_0000_0000_0000

The total command length must be less than 80 characters. The order of precedence is **left to right**. The Operation Priority Level (()) operators can be used; however, they cannot be nested.

[|]   Boolean Inclusive Or

		Product	**Rev**
Type	Operator(Bitwise)		
Syntax	See Below	6K	5.0
Units	n/a		
Range	n/a		
Default	n/a		
Response	n/a		
See Also	[=], [&], [~], [^], [<<], [>>], VAR, VARI, VARB		

The Boolean Inclusive Or (|) operator performs a logical OR on the two values to the left and right of the operator when used with the VAR or VARI command. The Boolean Inclusive Or (|) performs a bitwise OR on the two values to the left and right of the operator when used with the VARB command.

For a logical OR (using VAR or VARI), the possible combinations are as follows:

```
positive number | positive number                =  1
positive number | zero or a negative number      =  1
zero or negative number | positive number        =  1
zero or negative number | zero or negative number =  0
```

Example: VAR1=5 | –1
Result: VAR1=1

For a bitwise OR (using VARB), the value on the left side of the | operator has each of its bits *ORed* with the corresponding bit of the value on the right side of the operator. Each bit comparison will be composed of 9 possible combinations:

```
1 | 1 = 1              1 | X = 1
1 | 0 = 1              X | 1 = 1
0 | 1 = 1              0 | X = X
0 | 0 = 0              X | 0 = X
X | X = X
```

Example: VARB1=b1001 01X1 XX11 | b1000 1011 10
Response to VARB1 is *VARB1=1001_1111_1X11_XXXX_XXXX_XXXX_XXXX_XXXX

Example: VARB1=h1234 | hFAD31
Response to VARB1 is *VARB1=1111_0101_1111_1110_1000_0000_0000_0000

Example: VARB1=h23 | b1101 001X 001X 1X11
Response to VARB1 is *VARB1=1101_111X_001X_1X11_XXXX_XXXX_XXXX_XXXX

The total command length must be less than 80 characters. The order of precedence is **left to right**. The Operation Priority Level (()) operators can be used; however, they cannot be nested.

[^]   Boolean Exclusive Or

Type	`Operator (Bitwise)`		**Product**	**Rev**
Syntax	`See Below`		6K	5.0
Units	`n/a`			
Range	`n/a`			
Default	`n/a`			
Response	`n/a`			
See Also	`[=], [&], [~], [	], [<<], [>>], VAR, VARI, VARB`			

The Boolean Exclusive Or (`^`) operator performs a logical exclusive OR on the two values to the left and right of the operator when used with the `VAR` or `VARI` command. The Boolean Exclusive Or (`^`) performs a bitwise exclusive OR on the two values to the left and right of the operator when used with the `VARB` command.

For a logical exclusive OR (using `VAR` or `VARI`), the possible combinations are as follows:

positive number `^` positive number	`= 0`
positive number `^` zero or a negative number	`= 1`
zero or negative number `^` positive number	`= 1`
zero or negative number `^` zero or negative number	`= 0`

Example: `VAR1=5 ^ -1`
Result: `VAR1=1`

For a bitwise exclusive OR (using `VARB`), the value on the left side of the `^` operator has each of its bits exclusive *ORed* with the corresponding bit of the value on the right side of the operator. Each bit comparison will be composed of 9 possible combinations:

```
1 ^ 1 = 0          1 ^ X = X
1 ^ 0 = 1          X ^ 1 = X
0 ^ 1 = 1          0 ^ X = X
0 ^ 0 = 0          X ^ 0 = X
X ^ X = X
```

Example: `VARB1=b0000 1111 XXX1 ^ b10XX 10XX 10XX`
Response to `VARB1` is `*VARB1=10XX_01XX_XXXX_XXXX_XXXX_XXXX_XXXX_XXXX`

Example: `VARB1=h32FD ^ h6A`
Response to `VARB1` is `*VARB1=1010_0001_1111_1011_0000_0000_0000_0000`

Example: `VARB1=h7FFF ^ b1101 1111 0000 1101`
Response to `VARB1` is `*VARB1=0011_0000_1111_0010_XXXX_XXXX_XXXX_XXXX`

The total command length must be less than 80 characters. The order of precedence is **left to right**. The Operation Priority Level (`()`) operators can be used; however, they cannot be nested.

[~()]   Boolean Not

Type	`Operator (Bitwise)`		**Product**	**Rev**
Syntax	`See Below`		6K	5.0
Units	`n/a`			
Range	`n/a`			
Default	`n/a`			
Response	`n/a`			
See Also	`[=], [&], [^], [	], [<<], [>>], VAR, VARI, VARB`			

The Boolean Not (~) operator performs a logical NOT on the value immediately to its right when used with the `VAR` or `VARI` command. The Boolean NOT (~) performs a bitwise NOT on the value immediately to its right when used with the `VARB` command. Parentheses (`()`) are required.

For a logical NOT (using `VAR` or `VARI`), the possible combinations are as follows:

~ (positive number)	`=  0`
~ (zero or a negative number) `=`	`1`

Example: `VAR1=~(5)`   ; Result: `VAR1=0`
Example: `VAR1=~(-1)`   ; Result: `VAR1=1`

For a bitwise NOT (using VARB), each bit is *NOTed*.

> Example: VARB1=~(b0000 1000 1XX1)
> Response to VARB1 is *VARB1=1111_0111_0XX0_XXXX_XXXX_XXXX_XXXX_XXXX

> Example: VARB1=~(h32FD)
> Response to VARB1 is *VARB1=0011_1011_0000_0100_1111_1111_1111_1111

The total command length must be less than 80 characters. The order of precedence is **left to right**.

The Boolean Not (~) operator also has one additional use. It can be used to change the sign of the distance (D) command. (e.g., if the distance has the values *D+25000,+25000,+12000,-123000).

By issuing D~,~,~,~ the new values for distance would be *D-25000,-25000,-12000,+123000.

[<<]   Shift from R to L (Bit 32 to Bit 1)

Type	Operator (Bitwise	**Product**	**Rev**
Syntax	See Below	6K	5.0
Units	n/a		
Range	n/a		
Default	n/a		
Response	n/a		
See Also	[=], [&], [^], [	], [~], [>>], VAR, VARI, VARB		

The Shift R to L (<<) operator shifts a binary value from right to left (reducing its value) the number of bits specified. Zeros are shifted into the most significant bit locations. The number of bits to shift by is specified with the value immediately to the right of the (<<) operator, 32 maximum. The number of places to shift must be specified in either binary or hexadecimal format. (*The bits in the binary variable are displayed from 1 to 32, left to right, and shifting from right to left causes bits to be shifted from 32 to 1.*)

> Example: VARB1=b0000 1000 1XX1 << b01
> Response to VARB1 is *VARB1=0010_001X_X1XX_XXXX_XXXX_XXXX_XXX_XX00

> Example: VARB1=b1111 0000 1111 << b001
> Response to VARB1 is *VARB1=0000_1111_XXXX_XXXX_XXXX_XXXX_XXXX_0000

> Example: VARB1= h0000 E3 << hA
> Response to VARB1 is *VARB1=0000_0001_1111_0000_0000_0000_0000_0000

The total command length must be less than 80 characters. The order of precedence is **left to right**.

[>>]   Shift from L to R (Bit 1 to Bit 32)

Type	Operator (Bitwise)	**Product**	**Rev**
Syntax	See Below	6K	5.0
Units	n/a		
Range	n/a		
Default	n/a		
Response	n/a		
See Also	[=], [&], [^], [	], [~], [<<], VAR, VARI, VARB		

The Shift L to R (>>) operator shifts a binary value from left to right (increasing its value) the number of bits specified. Zeros are shifted into the least significant bit locations. The number of bits to shift by is specified with the value immediately to the right of the (>>) operator, 32 maximum. The number of places to shift must be specified in either binary or hexadecimal format. (*The bits in the binary variable are displayed from 1 to 32, left to right, and shifting from left to right causes bits to be shifted from 1 to 32.*)

> Example: VARB1=b0000 1000 1XX1 >> b01
> Response to VARB1 is *VARB1=0000_0010_001X_X1XX_XXXX_XXXX_XXXX_XXXX

> Example: VARB1=b1111 0000 1111 >> b001
> Response to VARB1 is *VARB1=0000_1111_0000_1111_XXXX_XXXX_XXXX_XXXX

> Example: VARB1= h45FA2 >> h4
> Response to VARB1 is *VARB1=0000_0010_1010_1111_0101_0100_0000_0000

The total command length must be less than 80 characters. The order of precedence is **left to right**.

[Send Response to Both Communication Ports

Type	Communication Interface	Product	Rev
Syntax	`<!> [ <command><field1>`	6K	5.0
Units	n/a		
Range	n/a		
Default	n/a		
Response	n/a		
See Also	BOT, PORT,], ECHO, EOL, EOT, LOCK		

The Send Response to All Ports (`[`) command is used to send the response from the command which follows it to all communication ports. If a syntax error occurs, an error message will be sent to both communication ports.

NOTE: COM1 refers to the "RS-232" or "ETHERNET" connector, and COM2 refers to the "RS-232/485" connector.

Example
```
[TER          ;Transfer TER Status to both serial ports
```

]   Send Response to Alternate Communication Port

Type	Communication Interface	Product	Rev
Syntax	`<!> ] <command><field1>`	6K	5.0
Units	n/a		
Range	n/a		
Default	n/a		
Response	n/a		
See Also	BOT, PORT, [, ECHO, EOL, EOT, LOCK		

The Send Response to Alternate Port (`]`) command is used to send the response from the command which follows it to the alternate port from the one selected. If a report back is requested from port COM1, the response will be sent out port COM2, and vice-versa. If a command is in a stored program, the report will be sent out the alternate port from the one selected by the PORT command. If a syntax error occurs an error message will be sent to the alternate port from the one selected.

NOTE: COM1 refers to the "RS-232" or "ETHERNET" connector, and COM2 refers to the "RS-232/485" connector.

Example
```
; *******************************************************************
; In this example, we place the "]TAS" statement in a program so that
; we can select the port (in this case, "PORT1" selects COM1) as a
; reference. Otherwise, executing "]TAS" outside of a program merely
; sends the response to whatever port you are not communicating through.
; *******************************************************************
DEF COM       ; Begin definition of program called "COM"
PORT1         ; Select COM1
TER           ; Transfer TER Status to port COM1
]TAS          ; Transfer TAS Status to port COM2
END           ; End program definition
```

A   Acceleration

Type	Motion		
		Product	**Rev**
Syntax	`<!><@><a>A<r>,<r>,<r>,<r>,<r>,<r>,<r>,<r>`	6K	5.0
Units	`r = units/sec/sec`		
Range	`0.00001 - 39,999,998 (depending on the SCLA scaling factor)`		
Default	`10.0000`		
Response	`A:   *A10.0000,10.0000,10.0000,10.0000 ...`		
	`1A:  *A10.0000`		
See Also	`[ A ], AA, AD, ADA, DRES, ERES, GO, MC, SCALE, SCLA, TSTAT`		

The Acceleration (`A`) command specifies the acceleration rate to be used upon executing the next go (`GO`) command.

> **UNITS OF MEASURE** and **SCALING**: refer to page 16.

The acceleration remains set until you change it with a subsequent acceleration command. Accelerations outside the valid range are flagged as an error, with a message `*INVALID DATA-FIELD x`, where x is the field number. When an invalid acceleration is entered the previous acceleration value is retained.

If the Deceleration (`AD`) command has not been entered, the acceleration (`A`) command will set the deceleration rate. Once the deceleration (`AD`) command has been entered, the acceleration (`A`) command no longer affects deceleration.

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

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

[A]   Acceleration Assignment

Type	Assignment or Comparison	**Product**	**Rev**
Syntax	See below	6K	5.0
Units	units/sec/sec		
Range	0.00001 – 39,999,998 (depending on the scaling factor)		
Default	n/a		
Response	n/a		
See Also	A, AA, AD, ADA, DRES, ERES, GO, SCALE, SCLA		

The acceleration assignment command is used to compare the programmed acceleration value to another value or variable, or to assign the current programmed acceleration to a variable.

Syntax: VARn=aA, where n is the variable number, and a is the axis number, or A can be used in an expression such as IF(1A<25ØØØ). When assigning the acceleration value to a variable, an axis specifier must always precede the assignment (A) command or it defaults to axis 1 (e.g., VAR1=1A). When making a comparison to the programmed acceleration, an axis specifier must also be used (e.g., IF(1A<2ØØØØ)). The (A) value used in any comparison, or in any assignment statement is the programmed (A) value.

UNITS OF MEASURE and **SCALING**: refer to page 16.

Example:
```
IF(2A<25000)      ; If the acceleration on axis 2 is less than 25000 units/sec/sec,
                  ; then do the statements between the IF and NIF
VAR1=2A*2         ; Variable 1 = acceleration of axis 2 times 2
A,(VAR1)          ; Set the acceleration on axis 2 to the value of variable 1
NIF               ; End the IF statement
```

AA   Average Acceleration

Type	Motion (S-Curve)	**Product**	**Rev**
Syntax	<!><@><a>AA<r>,<r>,<r>,<r>,<r>,<r>,<r>,<r>	6K	5.0
Units	r = units/sec/sec		
Range	0.00001 – 39,999,998 (depending on the scaling factor)		
Default	10.00 (trapezoidal profiling is default, where AA tracks A)		
Response	AA: *AA10.0000,10.0000,10.0000,10.0000		
	1AA: *1AA10.0000		
See Also	A, AD, ADA, SCALE, SCLA		

The Average Acceleration (AA) command allows you to specify the average acceleration for an S-curve motion profile. S-curve profiling provides smoother motion control by reducing the rate of change in acceleration and deceleration; this accel/decel rate of change is known as *jerk*. Refer to page 13 for details on S-curve profiling.

Scaling affects the average acceleration (AA) the same as it does for the maximum acceleration (A). Refer to page 16 for details on scaling.

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

Example:
```
; In this example, axis 1 executes a pure S-curve and takes 1 second
; to reach a velocity of 5 rps; axis 2 executes a trapezoidal profile
; and takes 0.5 seconds to reach a velocity of 5 rps.
SCALE0            ; Disable scaling
DEL proga         ; Delete program called proga
DEF proga         ; Begin definition of program called proga
@MA0              ; Select incremental positioning mode
@D40000           ; Set distances to 40,000 positive-direction steps
A10,10            ; Set max. accel to 10 rev/sec/sec (axes 1 and 2)
```

```
AA5,10            ; Set avg. accel to 5 rev/sec/sec on axis 1,
                  ; and 10 rev/sec/sec on axis 2
AD10,10           ; Set max. decel to 10 rev/sec/sec (axes 1 and 2)
ADA5,10           ; Set avg. decel to 5 rev/sec/sec on axis 1,
                  ; and 10 rev/sec/sec on axis 2
V5,5              ; Set velocity to 5 rps on axes 1 and 2
GO11              ; Execute motion on axes 1 and 2
END               ; End definition of program called proga
```

AD   Deceleration

			Product	Rev
Type	`Motion`		6K	5.0
Syntax	`<!><@><a>AD<r>,<r>,<r>,<r>,<r>,<r>,<r>,<r>`			
Units	`r = units/sec/sec`			
Range	`0.00001 - 39,999,998` (depending on the scaling factor)			
Default	`10.0000` (AD tracks A)			
Response	`AD:   *AD10.0000,10.0000,10.0000,10.0000 ...`			
	`1AD:  *AD10.0000`			
See Also	`[ A ], A, AA, ADA, DRES, ERES, GO, MC, SCALE, SCLA, TSTAT`			

The Deceleration (AD) command specifies the deceleration rate to be used upon executing the next go (GO) command.

> **UNITS OF MEASURE** and **SCALING**: refer to page 16.

The deceleration remains set until you change it with a subsequent deceleration command. Decelerations outside the valid range are flagged as an error, with a message *INVALID DATA-FIELD x, where x is the field number. When an invalid deceleration is entered the previous deceleration value is retained.

If the deceleration (AD) command has not been entered, the acceleration (A) command will set the deceleration rate. Once the deceleration (AD) command has been entered, the acceleration (A) command no longer affects deceleration. If the AD command is set to zero (AD∅), then the deceleration will once again track whatever the A command is set to.

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

Example:
```
SCALE1               ; Enable scaling
SCLA25000,25000,1,1  ; Set the acceleration scaling factor for axes 1 and 2 to
                     ; 25000 steps/unit, axes 3 and 4 to 1 step/unit
SCLV25000,25000,1,1  ; Set the velocity scaling factor for axes 1 and 2 to
                     ; 25000 steps/unit, axes 3 and 4 to 1 step/unit
@SCLD1               ; Set the distance scaling factor for all axes to 1 step/unit
DEL proga            ; Delete program called proga
DEF proga            ; Begin definition of program called proga
MA0000               ; Incremental index mode for all axes
MC0000               ; Preset index mode for all axes
A10,12,1,2           ; Set the acceleration to 10, 12, 1, and 2 units/sec/sec
                     ; for axes 1, 2, 3 and 4, respectively
AD1,1,1,2            ; Set the deceleration to 1, 1, 1, and 2 units/sec/sec for
                     ; axes 1, 2, 3 and 4, respectively
V1,1,1,2             ; Set the velocity to 1, 1, 1, and 2 units/sec for axes
                     ; 1, 2, 3 and 4, respectively
D100000,1000,10,100  ; Set the distance to 100000, 1000, 10, and 100 units for
                     ; axes 1, 2, 3 and 4, respectively
GO1100               ; Initiate motion on axes 1 and 2, 3 and 4 do not move
END                  ; End definition of program called proga
```

[AD]   Deceleration Assignment

Type	Assignment or Comparison	**Product**	**Rev**
Syntax	See below	6K	5.0
Units	units/sec/sec		
Range	0.00001 - 39,999,998 (depending on the scaling factor)		
Default	n/a		
Response	n/a		
See Also	[A], A, AA, AD, ADA, DRES, ERES, GO, SCALE, SCLA		

The deceleration assignment command is used to compare the programmed deceleration value to another value or variable, or to assign the current programmed deceleration to a variable.

Syntax: VARn=aAD where n is the variable number, and a is the axis number, or [AD] can be used in an expression such as IF(1AD<25ØØØ). When assigning the deceleration value to a variable, an axis specifier must always precede the assignment (AD)command or it defaults to axis 1 (e.g., VAR1=1AD). When making a comparison to the programmed deceleration, an axis specifier must also be used (e.g., IF(1AD<2ØØØØ)). The (AD) value used in any comparison, or in any assignment statement is the programmed (AD) value.

> **UNITS OF MEASURE** and **SCALING**: refer to page 16.

Example:
```
IF(2AD<25000)    ; If the deceleration on axis 2 is less than 25000 units/sec/sec,
                 ; then do the statements between the IF and NIF
VAR1=2AD*2       ; Variable 1 = deceleration of axis 2 times 2
AD,(VAR1)        ; Set the deceleration on axis 2 to the value of variable 1
NIF              ; End the IF statement
```

ADA   Average Deceleration

Type	Motion (S-Curve)	**Product**	**Rev**
Syntax	<!><@><a>ADA<r>,<r>,<r>,<r>,<r>,<r>,<r>,<r>	6K	5.0
Units	r = units/sec/sec		
Range	0.00001 - 39,999,998 (depending on the scaling factor)		
Default	10.00 (ADA tracks AA)		
Response	ADA: *ADA10.0000,10.0000,10.0000,10.0000 ...		
	1ADA: *1ADA10.0000		
See Also	A, AA, AD, SCALE, SCLA		

The Average Deceleration (ADA) command allows you to specify the average deceleration for an S-curve motion profile. S-curve profiling provides smoother motion control by reducing the rate of change in acceleration and deceleration; this accel/decel rate of change is known as *jerk*. Refer to page 13 for details on S-curve profiling.

Scaling affects the average acceleration (AA) the same as it does for the maximum acceleration (A). Refer to page 16 for details on scaling.

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

In the example below, axis 1 executes a pure S-curve and takes 1 second to return to zero velocity; axis 2 executes a trapezoidal profile and takes 0.5 seconds to return to zero velocity.

Example:
```
SCALE0          ; Disable scaling
DEL proga       ; Delete program called proga
DEF proga       ; Begin definition of program called proga
@MA0            ; Select incremental positioning mode
@D40000         ; Set distances to 40,000 positive-direction steps
```

```
A10,10              ; Set max. accel to 10 rev/sec/sec (axes 1 and 2)
AA5,10              ; Set avg. accel to 5 rev/sec/sec on axis 1,
                    ; and 10 rev/sec/sec on axis 2
AD10,10             ; Set max. decel to 10 rev/sec/sec (axes 1 and 2)
ADA5,10             ; Set avg. decel to 5 rev/sec/sec on axis 1,
                    ; and 10 rev/sec/sec on axis 2
V5,5                ; Set velocity to 5 rps on axes 1 and 2
GO11                ; Execute motion on axes 1 and 2
END                 ; End definition of program
```

ADDR   Multiple Unit Auto-Address

		Product	**Rev**
Type	`Controller Configuration`	6K	5.0
Syntax	`<!>ADDR<i>`		
Units	`i = axis number`		
Range	`0 to 99`		
Default	`0`		
Response	`ADDR: *ADDR0`		
See Also	`BAUD, E, PORT`		

The `ADDR` command automatically configures unit addresses for a daisy-chain or multi-drop. This command allows up to 99 units on a chain to be uniquely addressed.

The `ADDR` value is stored in non-volatile memory.

RS-232C Daisy Chain:

Sending `ADDRi` to the first unit in the chain sets its address to be `(i)`. The first unit in turn transmits `ADDR(i + 1)` to the next unit to set its address to `(i + 1)`. This continues down the daisy chain until the last unit of `(n)` daisy-chained units has its address set to `(i + n)`.

RS-485 Multi-Drop:

To use the `ADDR` command, you must address each unit individually before it is connected on the multi drop. For example, given that each product is shipped configured with address zero, you could set up a 4-unit multi-drop with the commands below, and then connect them in a multi drop:

1. Connect the unit that is to be unit #1 and transmit the `∅_ADDR1` command to it.
2. Connect the unit that is to be unit #2 and transmit the `∅_ADDR2` command to it.
3. Connect the unit that is to be unit #3 and transmit the `∅_ADDR3` command to it.
4. Connect the unit that is to be unit #4 and transmit the `∅_ADDR4` command to it.

   If you need to replace a unit in the multi drop, send the `∅_ADDRi` command to it, where "i" is the address you wish the new unit to have.

To send a 6K command from the master unit to a specific unit in the multi-drop, prefix the command with the unit address and an underscore (e.g., `3_OUT∅` turns off output #1 on unit #3). The master unit (if it is not a 6K product) may receive data from a multi-drop unit.

For more information on controlling multiple 6K Series controllers in an RS-232 daisy-chain or RS-485 multi-drop, refer to the *Programmer's Guide*.

Example:
```
ADDR1               ; Set the address of the first unit in the daisy-chain to 1
```

[AND]   And

Type	Operator(logical)	**Product**	**Rev**
Syntax	See below	6K	5.0
Units	n/a		
Range	n/a		
Default	n/a		
Response	n/a		
See Also	IF, [NOT], [OR], REPEAT, UNTIL, WAIT, WHILE		

The AND command is used in conjunction with the program flow control commands (IF, REPEAT, UNTIL, WHILE, WAIT). The AND command logically links two events. If each of the two events are true, and are linked with an AND command, then the whole statement is true. This fact is best illustrated by example.

Example 1:   IF(VAR1>Ø AND VAR2<3) : TPM : NIF

If variable 1 = 1 and variable 2 = 1, then the expression within the IF statement is true, and the commands between the IF and the NIF will be executed.

Example 2:   WHILE(VAR1=1 AND VAR2=2) : TPM : NWHILE

If variable 1 = 1 and variable 2 = 1, then the expression within the WHILE statement is false, and the commands between the WHILE and the NWHILE will not be executed.

To evaluate an expression (Expression 1 AND Expression 2 = Result) to determine if the whole expression is true, use the following rules:

TRUE AND TRUE = TRUE
TRUE AND FALSE = FALSE
FALSE AND TRUE = FALSE
FALSE AND FALSE = FALSE

[ANI]   Analog Input Value

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	ANIRNG, [FB], [PANI], SFB, TANI, TFB, TPANI		

Use the ANI operator to assign the voltage level present at one of the analog inputs (ANI) to a variable, or to make a comparison against another value. The ANI value is measured in volts and does not reflect the effects of distance scaling (SCLD), position offset (PSET), or commanded direction polarity (CMDDIR). To assign/compare the ANI input value, as affected by SCLD, PSET, and CMDDIR, use the PANI command or the FB command.

The ANI value is derived from the voltage applied to the corresponding analog input and ground. The analog value is determined from a 12-bit analog-to-digital converter. Under the default ANI voltage range, set with ANIRNG, the range of the ANI operator is -10.000VDC to +10.000VDC (see ANIRNG command for optional voltage ranges).

Syntax: VARn=<B>ANI.i where "n" is the variable number, "<B>" is the number of the I/O brick, and "i" is I/O brick address where the analog input resides; or ANI can be used in an expression such as IF(1ANI.2=2.3). If no brick identifier (<B>) is provided, it defaults to 1. To understand the I/O brick addressing convention, refer to page 6.

Example:
```
VAR2=3ANI.2     ; Voltage value at analog input 2 on I/O brick 3 is assigned
                ; to variable 2
IF(1ANI.1<8.2)  ; If voltage value at analog input 1 on brick 1 < 8.2V, do the
                ; commands between the IF statement and the NIF statement.
TREV            ; Transfer revision level
NIF             ; End if statement
```

ANIEN   Analog Input Enable

		Product	Rev
Type	Inputs	6K	5.0
Syntax	To enable only: <!><B>ANIEN<.i>=<E> To override only: <!><B>ANIEN<.i>=<r>		
Units	B = I/O brick number i = input location on I/O brick "B" E = Enable r = volts		
Range	B = 1-8 i = 1-32 (dependent on I/O brick configuration) r = -10.000 to +10.000 (voltage override value)		
Default	E (enabled)		

Response	2ANIEN: *2ANIENx,x,x,x,x,x,x,x (SIM slot 1)
	          x,x,x,x,x,x,x,x (SIM slot 2)
	          E,E,E,E,E,E,E,E (SIM slot 3)
	          x,x,x,x,x,x,x,x (SIM slot 4)		

See Also	[ANI], ANIFB, ANIMAS, ANIRNG, FOLMAS, TANI, TIO		

The Analog Input Enable (ANIEN) command enables or disables specific analog inputs. The default state for each input is the enabled condition. ANIEN can also be used to set analog inputs to specific override voltage levels. To disable an analog input, set an override voltage of 0.

Performance: The rate at which the controller samples each analog input depends on how many are enabled on the SIM; each enabled analog input adds 2 ms to the sample rate for all analog inputs on the SIM. For example, if 4 of the 8 analog inputs on a SIM are enabled, the sample rate for any specific input on the same SIM is 8 ms (4 inputs x 2 ms). Disabling input channels increases the performance of the remaining channels; this is important if an input channel is to be used as a servo feedback source (ANIFB and SFB selection) or Following source (ANIMAS and FOLMAS selection).

Example:
```
1ANIEN.9=E,E   ; Enable the 1st & 2nd analog input in SIM slot 2 (I/O
               ; locations 9 & 10) on I/O brick 1.
1ANIEN.11=E,2.5 ; Override the 3rd analog input in SIM slot 2 (I/O location 11)
               ; on I/O brick 1 with a voltage of 2.4 volts.
2ANIEN         ; Check status of analog inputs on I/O brick 2. As an example,
               ; a response of *2ANIENx,x,x,x,x,x,x,x
               ;                        x,x,x,x,x,x,x,x
               ;                        E,E,E,E,E,E,E,E
               ;                        x,x,x,x,x,x,x,x
               ; indicates that an analog input SIM is installed in slot 3 of
               ; I/O brick 2 and all eight channels are enabled ("E").
```

ANIFB   Assign Analog Inputs as Axis Feedback

		Product	Rev
Type	Controller Configuration	6K	5.0
Syntax	<!>ANIFB<B-i>,<B-i>,<B-i>,<B-i>,<B-i>,<B-i>,<B-i>,<B-i>		
Units	B = I/O brick number i = input location on I/O brick "B"		
Range	B = 1-8 i = 1-32 (dependent on I/O brick configuration)		
Default	0-0 (No assignment)		
Response	ANIFB: *ANIFB1-1,1-9,0,0,0,0,0,0		
See Also	[ANI], ANIEN, ANIRNG, SFB, TANI, TIO		

The ANIFB command determines which analog (ANI) inputs to use as feedback sources for specific axes when ANI feedback is selected by the SFB command. The ANIFB command only has an effect if ANI feedback is selected by a subsequent SFB command (ANIFB command must be issued <u>before</u> the SFB command).

Example
```
ANIFB,,,,,4-17  ; Select the 1st analog input channel in SIM slot 3
                ; (I/O location 17) of I/O brick 4 to be used as
                ; feedback for axis 6
SFB,,,,,2       ; Select analog input feedback for axis 6
```

ANIMAS   Assign Analog Inputs to Axes

		Product	**Rev**
Type	Following	6K	5.0
Syntax	<!>ANIMAS<B-i>,<B-i>,<B-i>,<B-i>,<B-i>,<B-i>,<B-i>,<B-i>		
Units	B = I/O brick number		
	i = input location on I/O brick "B"		
Range	B = 1-8		
	i = 1-32 (dependent on I/O brick configuration)		
Default	0-0 (No assignment)		
Response	ANIMAS: *ANIMAS1-1,1-9,0,0,0,0,0,0		
See Also	ANIEN, FOLMAS		

The ANIMAS command assigns an analog input channel to a specific Following master axis for use when an ANI master is selected with the FOLMAS command. The ANIMAS command only has an effect if an analog input Following master is selected with a subsequent FOLMAS command (ANIMAS command must be issued <u>before</u> the FOLMAS command).

Example
```
ANIMAS,,,,,4-17      ; Select the first analog input channel in SIM slot 3
                     ; (I/O location 17) of I/O brick 4 to be used for
                     ; master axis 6
FOLMAS62,62          ; Define axes 1 and 2 to be followers of the analog input
                     ; selected for master axis 6
```

ANIRNG   Analog Input Voltage Range

		Product	**Rev**
Type	Controller Configuration	6K	5.0
Syntax	<!><B>ANIRNG<.i><=i>		
Units	B = I/O brick number		
	1st i = input location on I/O brick "B"		
	2nd i = voltage range selector number		
Range	B = 1-8		
	1st i = 1-32 (dependent on I/O brick configuration)		
	2nd i = 1 (0 to +5VDC),		
	2 (-5 to +5VDC),		
	3 (0 to +10VDC), or		
	4 (-10 to +10VDC)		
Default	4 (range is set to -10 to +10VDC)		
Response	2ANIRNG: *2ANIRNGx,x,x,x,x,x,x,x		
	4,4,4,4,4,4,4,4		
	x,x,x,x,x,x,x,x		
	x,x,x,x,x,x,x,x		
	2ANIRNG.9: *4		
See Also	[ANI], ANIEN, ANIFB, JOYCDB, JOYCTR, JOYEDB, [PANI],		
	SCLA, SCLV, SFB, TANI, TIO, TPANI		

Use the ANIRNG command to select voltage ranges for specific analog inputs on the expansion I/O brick connected to your 6K product. The default range for all analog inputs -10VDC to +10VDC.

Be aware that changing the analog input voltage range affects these settings:

ANIRNG Setting	Voltage Range	Counts/volt resolution (see PANI & TPANI)	Calculation for minimum accel **	Calculation for maximum accel **	Calculation for maximum velocity **
1	0 to +5VDC	819	$\dfrac{0.819}{SCLA}$	$\dfrac{819,000}{SCLA}$	$\dfrac{819,000}{SCLA}$
2	-5 to +5VDC	410	$\dfrac{0.410}{SCLA}$	$\dfrac{410,000}{SCLA}$	$\dfrac{410,000}{SCLA}$
3	0 to +10VDC	410	$\dfrac{0.410}{SCLA}$	$\dfrac{410,000}{SCLA}$	$\dfrac{410,000}{SCLA}$
4	-10 to +10VDC	205	$\dfrac{0.205}{SCLA}$	$\dfrac{205,000}{SCLA}$	$\dfrac{205,000}{SCLA}$

** These calculations are for servo axes using an analog input as it's position feedback source (see ANIFB and SFB).

Example
```
2ANIRNG.9=3      ; For the 1st analog input on SIM2 of I/O brick 2,
                 ; select a voltage range of 0 to +10VDC
```

[AS]   Axis 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], GOWHEN, INDUST, SMPER, TAS, TASF, TRGFN, TSTAT, VARB			

Use the AS operator to assign the axis status bits for a specific axis 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 that the axis status is being compared against. The binary value itself must only contain ones, zeros, or Xs (1, Ø, 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 axis status is being compared against. The hexadecimal value itself must only contain the letters A through F, and the numbers Ø through 9. When using AS, an axis specifier must always proceed it, or else it will default to axis 1. Valid axis specifiers are 1, 2, 3, or 4 (1AS, 2AS, 3AS, or 4AS). The function of each axis status bit is shown below. An "x" identifies products to which the function is applicable.

Bit # (left to right)	**Function** (1/Ø)
1	Moving/Not Moving. This bit is set only when motion is <u>commanded</u> on the axis. The motor may still be "moving" (e.g., due to end-of-move settling).
2	Negative/positive-direction
3	Accelerating/Not Accelerating. This bit does not indicate deceleration (bit is set to 0 during decel); to check if the axis is decelerating, the state of AS bits 1, 3 and 4 should be: AS1x00.
4	At Velocity/Not at Velocity
5	Home Successful (HOM) (YES/NO)
6	Absolute/Incremental (MA1/MA0)
7	Continuous/Preset (MC1/MC0)
8	Jog Mode/Not Jog Mode (JOG)
9	Joystick Mode/Not Joystick Mode (JOY1/JOY0)
10	RESERVED
11	RESERVED
12	Stall Detected (YES/NO). This bit is not usable until Stall Detect is enabled with ESTALL1 command.
13	Drive Shut Down (YES/NO)
14	Drive Fault occurred (YES/NO). A drive fault cannot be detected (this bit is always 0) until the drive fault input check is enabled with DRFEN1. <u>Note</u>: ASX bit 4 reports the hardware state of the drive fault input, regardless of DRFEN or DRIVE.
15	Positive-direction Hardware Limit Hit (YES/NO)
16	Negative-direction Hardware Limit Hit (YES/NO)
17	Positive-direction Software Limit Hit (YES/NO)
18	Negative-direction Software Limit Hit (YES/NO)
19	RESERVED
20	RESERVED
21	RESERVED
22	RESERVED
23	Position Error Exceeded (SMPER) (YES/NO). Servo axes only.
24	In Target Zone (defined with STRGTD & STRGTV) (YES/NO). Servo axes only. This bit is set only after the *successful completion* of a move (if STRGTD and STRGTV criteria have been satisfied). This bit is usable even if the Target Zone mode is not enabled (STRGTE0).

Bit # (left to right)	Function (1/∅)
25	Target Zone Timeout occurred (STRGTT) (YES/NO). Servo axes only.
26	Change in motion is suspended pending GOWHEN (YES/NO). This bit is cleared if the GOWHEN condition is true, or if STOP (!S) or KILL (!K or ^K) is executed.
27	RESERVED
28	Registration move initiated by trigger since last GO command. This bit is cleared with the next GO command.
29	RESERVED
30	Pre-emptive (OTF) GO or Registration profile not possible
31	RESERVED
32	RESERVED

Syntax: VARBn=aAS where n is the binary variable number and a is the axis identifier, or AS can be used in an expression such as IF(1AS=b11∅1), or IF(1AS=h7F). If it is desired to assign only one bit of the axis status value 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 axis status bit (e.g., VARB1=1AS.12 assigns axis 1 status bit 12 to binary variable 1).

Example:
```
VARB1=1AS            ; Axis status for axis 1 assigned to binary variable 1
VARB2=1AS.12         ; Axis 1 status bit 12 assigned to binary variable 2
VARB2                ; Response, if bit 12 is set to 1, is
                     ; "*VARB2=XXXX_XXXX_XXX1_XXXX_XXXX_XXXX_XXXX_XXXX"
IF(4AS=b111011X11)   ; If the axis 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(2AS=h7F00)        ; If the axis 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
```

[ASX]　　　　　Extended Axis 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	EFAIL, TASX, TASXF, [AS], TAS, TASF, VARB		

The Extended Axis Status (ASX) command is used to assign the axis status bits for a specific axis 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 that the axis status is being compared against. The binary value itself must only contain ones, zeros, or Xs (1, Ø, 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 axis status is being compared against. The hexadecimal value itself must only contain the letters A through F, and the numbers Ø through 9. An "x" identifies products to which the function is applicable.

Bit Assignment (left to right)	**Function** (1 = yes, Ø = no)
1-3	RESERVED
4 *	Drive Fault Input Active
5 **	Encoder Failure
6	Z-channel state (1 = active, Ø = inactive)
7-32	RESERVED

* Bit #4 indicates the current hardware state of the drive fault input, even in the factory default power-up state —the drive is disabled (see DRIVE command) and the drive fault input is disabled (see DRFEN command).

** Bit #5 requires the Encoder Failure detection be enabled for the particular axis (see EFAIL command); this bit is cleared with the EFAILØ command.

Syntax:　VARBn=ASX where n is the binary variable number, or ASX can be used in an expression such as IF(ASX=b11ØØ), or IF(ASX=h7Ø). If it is desired to assign only one bit of the axis status value 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 axis status bit (e.g., VARB1=ASX.3 assigns axis 1 status bit 3 to binary variable 1).

Example:
```
VARB1=ASX            ; Extended Axis status for axis 1 assigned to
                     ; binary variable 1
VARB2=ASX.3          ; Extended Axis 1 status bit 3 assigned to
                     ; binary variable 2
VARB2                ; Response if bit 3 is set to 1:
                     ; "*VARB2=XX1X_XXXX_XXXX_XXXX_XXXX_XXXX_XXXX_XXXX"
IF(ASX=b101XXXXX)    ; If the extended axis status for axis 1 contains 1's
                     ; for bits 1 and 3, and a 0 for bit location 2, do the
                     ; IF statement
TREV                 ; Transfer revision level
NIF                  ; End if statement
```

[ATAN()]   Arc Tangent

|			
Type	Operator (Trigonometric)	**Product**	**Rev**
Syntax	VARi=ATAN(r)	6K	5.0
Units	r = real number		
Range	0.00000 to ±999,999,999		
Default	none		
Response	n/a		
See Also	[=],[COS],[PI],RADIAN,[SIN],[TAN],VAR		

This Arc Tangent (ATAN) operator is used to calculate the inverse tangent of a real number. If "a" and "b" are coordinates of a point on a circle of radius "r", then the angle of measure "θ" can be defined by the equation:

$$\theta = \arctan \frac{a}{b} .$$

The result of the ATAN command will either be in degrees or radians, depending on the RADIAN command.

To convert radians to degrees, use the formula: $360° = 2\pi$ radians.
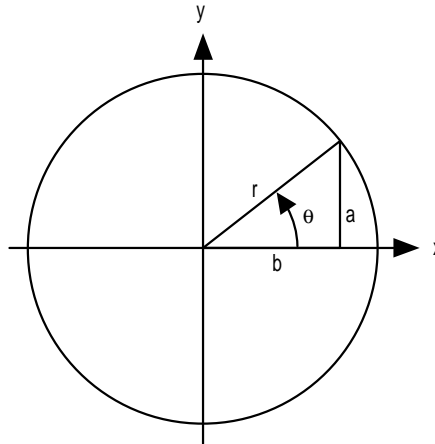
$$\sin \theta = \frac{a}{r}$$

$$\cos \theta = \frac{b}{r}$$

$$\tan \theta = \frac{a}{b}$$

Syntax: VARi=ATAN(r) where i is the variable number and r is a real number value. Parentheses (()) must be used with the ATAN command. **The result will be specified to 2 decimal places in either radians or degrees.**

Example:
```
RADIAN1              ; Enable radian mode
VAR1=ATAN(0.75)      ; Set variable 1 equal to the inverse tangent of 0.75 radians
```

AXSDEF   Axis Definition

Type	Controller Configuration	**Product**	**Rev**
Syntax	<!><@>AXSDEF<b><b><b><b><b><b><b><b>	6K	5.0
Units	n/a		
Range	b = 0 (stepper), 1 (servo), or X (don't change)		
Default	1 (servo)		
Response	AXSDEF: *11111111		
See Also	DRIVE		

The Axis Definition (AXSDEF) command identifies the type of drive (servo or stepper) to which the controller axis is connected. <u>The drive must be disabled (DRIVE0) for the AXSDEF command to function properly.</u> Stepper drives receive their positioning information via step and direction signals. Servo drives receive their positioning commands via a ±10 volt signal. The AXSDEF setting is automatically saved in battery backed RAM.

The value of AXSDEF disables command fields that are not appropriate for that type of drive. For example, an axis configured as a stepper cannot be affected by a Servo Proportional Gain (SGP) command. The report back of non-applicable commands contains "–" in the field for that axis.

AXSDEF0 — Stepper Only Commands:	
DRES	FMAXA
ENCCNT	FMAXV
ESDB	PULSE
ESK	
ESTALL	

AXSDEF1 — Servo Only Commands:			
ANIFB	KDRIVE	PER	SFB
SGP	SGI	SGV	SGVF
SGAF	SGILIM	SOFFS	SMPER
SGSET	SGENB	STRGTD	STRGTE
STRGTT	STRGTV	TFB	TGAIN
TPER	TSGSET	TSTLT	

NOTE: If you change the axis definition, be sure to verify or set all motion settings and scaling values to achieve the expected performance.

BAUD   Baud Rate

Type	Communication Interface	**Product**	**Rev**
Syntax	BAUD<i>	6K	5.0
Units	i = Baud rate		
Range	i = 1200, 2400, 4800, 9600, 19200, 38400, or 115200		
Default	9600		
Response	BAUD *BAUD9600		
See Also	ADDR, E, PORT		

BAUD establishes the baud rate for the "RS-232" (COM1) or the "RS-232/485" (COM2) serial port, as selected by the last PORT command. The default is 9600 baud (BAUD9600). The BAUD setting is automatically saved in battery backed RAM. **NOTE**: Changing the baud rate for the currently used port will result in a loss of communication until the baud rate of the terminal is changed accordingly.

Example:
```
PORT2           ; Select COM2 ("RS-232/485") port
BAUD38400       ; Set the baud rate for COM2 to 38400 baud
```

BOT   Beginning of Transmission Characters

Type	Communication Interface	**Product**	**Rev**
Syntax	<!>BOT<i>,<i>,<i>	6K	5.0
Units	n/a		
Range	i = 0 – 256		
Default	0,0,0		
Response	BOT: *BOT0,0,0		
See Also	EOT, ERROK, ERRBAD, PORT, DRPCHK,EOL,], [		

The Beginning of Transmission Characters (BOT) command designates the characters to be placed at the beginning of every response. Up to 3 characters can be placed before the first line of a multi-line response, or before all single-line responses. The characters are designated with their ASCII equivalent. 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. Note that ASCII 256 means ØØ is transmitted.

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

Example:
```
BOT13,10,26     ; Place a carriage return, line feed, and Ctrl-Z before
                ; the first line of a multi-line response, and before
                ; all single line responses
```

BP   Set a Program Break Point

Type	Program Flow Control or Program Debug Tool	**Product**	**Rev**
Syntax	<!>BP<i>	6K	5.0
Units	i = break point number		
Range	1 – 32		
Default	n/a		
Response	n/a		
See Also	BREAK, C, HALT, K, S, [SS], TSS		

The Break Point (BP) command allows the programmer to set a place in the program where command processing will halt and a message will be transmitted to the PC. There are 32 break points available, BP1 to BP32, all transmitting the message *BREAKPOINT NUMBER x<cr> where x is the break point number.

After halting at a break point, command processing can be resumed by issuing a continue (!C) command.

The break point command is useful for stopping a program at specific locations in order to test status for debugging or other purposes.

Example:
```
DEF prog1          ; Begin definition of program named prog1
D50000,1000        ; Set distance to 50000 units on axis 1, and 1000 units on axis 2
MA1100             ; Absolute mode for axes 1 and 2
GO1100             ; Initiate motion on axes 1 and 2
IF(1PC>40000)      ; Compare axis 1 commanded position to 40000
BP1                ; If the motor position is > 40000 units, set break point #1
NIF                ; End IF statement
D80000,2000        ; Set distance to 80000 units on axis 1, and 2000 units on axis 2
GO1100             ; Initiate motion on axes 1 and 2
BP2                ; Set break point #2
END                ; End program definition
RUN prog1          ; Execute program prog1
```

If the `IF` statement evaluates true, the message `*BREAKPOINT NUMBER 1` will be transferred out. A `!C`
command must be issued before processing will continue. Once processing has continued, the second break
point command will be encountered, again the message `*BREAKPOINT NUMBER 2` will be transferred out,
and processing of commands will pause until a second `!C` command is received.

BREAK   Terminate Program Execution

Type	Program Flow Control	**Product**	**Rev**
Syntax	`<!>BREAK`	6K	5.0
Units	n/a		
Range	n/a		
Default	n/a		
Response	n/a		
See Also	BP, C, GOSUB, HALT, K, S		

The `BREAK` command terminates program execution when processed. This command allows the user to
terminate a program based upon a condition, or at any other particular point in the program where it is
necessary to end the program. If the program terminated was called from another program, control will be
passed to the calling program. This command is useful when debugging a program.

To terminate all program processing, use the `HALT` command.

Example:
```
DEF prog1          ; Define a program called prog1
GO1000             ; Initiate motion on axis 1
GOSUB prog2        ; Gosub to subroutine named prog2
GO0100             ; Initiate motion on axis 2
END                ; End program definition
DEF prog2          ; Define a program called prog2
GO1110             ; Initiate motion on axes 1, 2, and 3
IF(IN=b1X0)        ; IF condition is: status of trigger input 1 is
                   ; active (1) and trigger input 3 is inactive (0)
BREAK              ; If condition is true break out of program
ELSE               ; Else part of if condition
TPE                ; If condition does not come true, transfer position of
                   ; all encoders
NIF                ; End If statement
END                ; End program definition
RUN prog1          ; Execute program prog1
;
; Upon completion of motion on axis 1, subroutine prog2 is called. If inputs 1
; and 3 are in the correct state when the subroutine is entered, the subroutine
; will be terminated and returned to prog1, where motion on axis 2 will be
; initiated.
```

C   Continue Command Execution

Type	Program Flow Control			**Product**	**Rev**
Syntax	`!C`			6K	5.0
Units	n/a				
Range	n/a				
Default	n/a				
Response	n/a				
See Also	BP, COMEXR, COMEXS, INFNC, PS, S				

The Continue (`!C`) command ends a pause state (`PS`), a break point (`BP`) condition, or a stopped (`S`) condition. When the controller is in a paused state or at a break point, no commands from the command buffer are executed. All immediate commands, however, are still processed. By sending a `!C` command, command processing will resume, starting with the first command after the `PS` command or the `BP` command. If a stop (`S`) command has been issued, motion and command processing can be resumed by issuing a `!C` command, only if `COMEXS` has been enabled.

Example:
```
PS              ; Stop execution of command buffer until !C command
MA0XXX          ; Incremental mode for axis 1
D10000          ; Set distance to 10000 units on axis 1
GO1000          ; Initiate motion on axis 1
D,20000         ; Set distance to 20000 units on axis 2
GO0100          ; Initiate motion on axis 2
```

No buffered commands after the `PS` command will be executed until a `!C` command is received.

```
!C              ; Restart execution of command buffer
DEF prog1       ; Begin definition of program named prog1
D50000,1000     ; Set distance to 50000 units on axis 1, & 1000 units on axis 2
MA00            ; Set axes 1 and 2 to the incremental mode
GO11            ; Initiate motion on axes 1 and 2
IF(VAR1>6)      ; Compare VAR1>6
BP1             ; If the motor position is > 50000 units, set break point #1
NIF             ; End IF statement
GO11            ; Initiate motion on axes 1 and 2
BP2             ; Set break point #2
END             ; End program definition
RUN prog1       ; Execute program prog1
```

If the `IF` statement evaluates true, the message `BREAKPOINT NUMBER 1` will be transferred out. A `!C` command must be issued before processing will continue. Once processing has continued, the second break point command will be encountered, again the message `BREAKPOINT NUMBER 2` will be transferred out, and processing of commands will pause until a second `!C` command is received.

```
COMEXS1         ; Enable command processing on stop
D50000,1000     ; Set distance to 50000 units on axis 1, & 1000 units on axis 2
GO1100          ; Initiate motion on axes 1 and 2
!S              ; Stop motion on all axes
```

When the 6K Series product processes the `!S` command, motion on all axes will be stopped. If the desired distance has not been reached, motion can be resumed by issuing the `!C` command. If motion and command processing are to stop, a Kill (`!K`) command can be issued.

CMDDIR   **Commanded Direction Polarity**

Type	Controller Configuration	**Product**	**Rev**
Syntax	<@><a>CMDDIR<b><b><b><b><b><b><b><b>	6K	5.0
Units	b = polarity bit		
Range	0 (normal polarity), 1 (reverse polarity) or X (don't change)		
Default	0		
Response	CMDDIR *CMDDIR0000_0000		
	1CMDDIR *1CMDDIR0		
See Also	[AS], DRIVE, ENCPOL, [FB], [PANI], [PCE], [PE], [PER], PSET, SFB, TAS, TFB, TPANI, TPCE, TPE, TPER		

The CMDDIR command allows you to reverse the direction that the controller considers to be the "positive" direction; this also reverses the polarity of the counts from the feedback devices. Thus, using the CMDDIR command, you can reverse the referenced direction of motion without the need to (a) change the connections to the drive and the feedback device, or (b) change the sign of all the motion-related commands in your program.

NOTES

- SERVO AXES: Before changing the commanded direction polarity, make sure there is a direct correlation between the commanded direction and the direction of the feedback source counts (i.e., a positive commanded direction from the controller must result in positive counts from the feedback device). Refer to the ENCPOL command description for information on changing encoder polarity.
- Once you change the commanded direction polarity, you should swap the end-of-travel limit connections to maintain a positive correlation with the commanded direction.

The CMDDIR command is automatically saved in non-volatile memory.

The CMDDIR command cannot be executed while motion is in progress or while the drive/valve is enabled. For example, you could wait for motion to be complete (indicated when AS bit #1 is a zero) and then use the DRIVE command to disable the appropriate axis before executing the CMDDIR command.

COMEXC   **Continuous Command Processing Mode**

Type	Command Buffer Control	**Product**	**Rev**
Syntax	<!>COMEXC<b>	6K	5.0
Units	b = 0, 1 or X		
Range	0 = Disable, 1 = Enable, X = don't change		
Default	0		
Response	COMEXC: *COMEXC0		
See Also	[!], A, AA, AD, ADA, COMEXL, COMEXS, D, ERRORP, FOLRD, FOLRN, GO, GOWHEN, MA, MC, V		

This command enables (COMEXC1) or disables (COMEXCØ) Continuous Command Execution Mode. Normally, when a motion command is received, command processing is temporarily paused until the motion is complete. In continuous command execution mode, however, command processing continues while motion is taking place. **NOTE**: Command processing will be slower and **some** motion parameters cannot be changed while motion is in progress; for a complete list of motion parameters that cannot be changed while motion is in progress, refer to the Restricted Commands During Motion section in Chapter 1 of the *Programmer's Guide*.

The Continuous Command Processing Mode is useful in the following situations:

- When trying to check the status of inputs while the 6K Series product is commanding motion.

- Performing calculations ahead of time, possibly decreasing cycle time.

- Executing buffered on-the-fly acceleration (A, AA), and deceleration (AD, ADA), distance (D), positioning mode (MA & MC), Following ratio (FOLRD & FOLRN), and velocity (V) changes. (The buffered A, AA, AD, ADA, D, FOLRD, FOLRN, MA, MC, or V change can be executed only with a buffered Go (GO) command.) For more information about on-the-fly motion changes, refer to the *Programmer's Guide*.

- Pre-processing the next move while the current move is in progress (see CAUTION note below). This reduces the processing time for the subsequent move to only a few microseconds.

CAUTION: Avoid Executing Moves Prematurely

With continuous command execution enabled (COMEXC1), if you wish motion to stop before executing the subsequent move, place a WAIT(AS.1=bØ) statement before the subsequent GO command. If you wish to ensure the load settles adequately before the next move, use the WAIT(AS.24=b1) command instead (this requires you to define end-of-move settling criteria — see STRGTE command or *Programmer's Guide* for details).

Example:
```
VAR1=2000       ; Set variable 1 = 2000
VAR2=0          ; Set variable 2 = 0
COMEXC1         ; Enable continuous command execution mode
L50             ; Loop 50 times
D50000,(VAR1)   ; Set distance to 50000 units for axis 1, VAR1 value for axis 2
GO1100          ; Initiate motion on axes 1 and 2
; Normally at this point, the 6K controller would wait for the motion on axes
; 1 & 2 to complete before processing the next command. However, with continuous
; command execution enabled (COMEXC1), processing will continue with the
; statements that follow.
REPEAT          ; Beginning of REPEAT..UNTIL() expression
IF(IN.1=b1)     ; Check for onboard input #1 (trigger A for axis 1)
                ; becoming active
VAR1=VAR1+10    ; If it does, increase variable 1 by 10
VAR2=1          ; Variable 2 is used as a flag
NIF             ; End IF statement
UNTIL(MOV=b0 OR VAR2=5)    ; Exit REPEAT loop if variable 2 equals 5 or if
                ; motion is complete on axis 1
VAR2=0          ; Reset flag value, variable 2 = 0
LN              ; End loop
COMEXC0         ; Disable continuous command mode
```

On-the-fly Velocity, Acceleration and Deceleration Change Example:
```
DEF vsteps      ; Begin definition of program vsteps
COMEXC1         ; Enable continuous command execution mode
MC1             ; Set axis 1 mode to continuous
A10             ; Set axis 1 acceleration to 10 rev/sec/sec
V1              ; Set axis 1 velocity to 1 rps
GO1             ; Initiate axis 1 move (Go)
WAIT(1VEL=1)    ; Wait for motor to reach continuous velocity
T3              ; Time delay of 3 seconds
A50             ; Set axis 1 acceleration to 50 rev/sec/sec
V10             ; Set axis 1 velocity to 10 rps
GO1             ; Initiate axis 1 move (Go)
T5              ; Time delay of 5 seconds
S1              ; Initiate stop of axis 1 move
WAIT(MOV=b0)    ; Wait for motion to completely stop on axis 1
COMEXC0         ; Disable continuous command execution mode
END             ; End definition of program vsteps
```

COMEXL   Continue Execution on Limit

Type	Command Buffer Control	**Product**	**Rev**
Syntax	`<!><@><a>COMEXL<b><b><b><b><b><b><b><b>`	6K	5.0
Units	b = 0, 1 or X		
Range	0 = Disable, 1 = Enable, X = don't change		
Default	0		
Response	COMEXL: *COMEXL0000_0000		
	1COMEXL: *1COMEXL0		
See Also	COMEXC, COMEXS, ERROR, LH, LHLVL, LS		

This command determines whether the command buffer will be saved upon hitting a hardware end-of-travel limit (LH), or a soft limit (LS). If save command buffer on limit is enabled (COMEXL1), then all commands following the command currently being executed will remain in the command buffer when a limit is hit. If save command buffer on limit is disabled (COMEXLØ), then every command in the buffer will be discarded, and program execution will be terminated.

Example:
```
COMEXL0010          ; Save the command buffer only if the limit on axis 3 is hit.
                    ; Hitting a limit on any other axis will dump the command buffer.
```

COMEXR   Continue Motion on Pause/Continue Input

Type	Command Buffer Control	**Product**	**Rev**
Syntax	`<!>COMEXR<b>`	6K	5.0
Units	b = 0, 1 or X		
Range	0 = disable, 1 = enable, X = don't change		
Default	0		
Response	COMEXR: *COMEXR0		
See Also	C, COMEXS, INFNC, LIMFNC		

The Continue Motion on Pause/Continue (COMEXR) command determines the functionality of programmable inputs defined as pause/continue inputs with the INFNCi-E or LIMFNCi-E command. In both cases, when the input is activated (exception: an axis-specific step input will not dump the buffer), the current command being processed will be allowed to finish executing.

COMEXRØ: Upon receiving a pause input, only program execution is paused; any motion in progress will continue to its predetermined destination. Releasing the pause input or issuing a !C command will resume program execution.

COMEXR1:   Upon receiving a pause input, both motion and program execution will be paused; the motion stop function is used to halt motion. *After motion stops*, you can release the pause input or issue a !C command to resume motion and program execution.

Example:
```
COMEXR1            ; Allow both motion and program execution to be paused upon
                   ; receiving a pause input
2INFNC1-E          ; Define input 1 on I/O brick 2 as a pause/continue input
```

COMEXS   Continue Execution on Stop

Type	`Command Buffer Control`	**Product**	**Rev**
Syntax	`<!>COMEXS<i>`	6K	5.0
Units	`i = function identifier`		
Range	`0, 1, or 2`		
Default	`0`		
Response	`COMEXS:  *COMEXS0`		
See Also	`COMEXC, COMEXL, COMEXR, INFNC, LIMFNC, S`		

The Continue Execution on Stop (`COMEXS`) command determines whether the command buffer will be saved upon receiving a Stop command (`!S` or `!S1111`) or an external stop input (`INFNCi-D` or `LIMFNCi-D`).

`COMEXSØ`:   Upon receiving a stop input or Stop command, motion will decelerate at the preset `AD`/`ADA` value, every command in the buffer will be discarded (exception: an axis-specific stop input will not dump the buffer), and program execution will be terminated.

`COMEXS1`:   Upon receiving a stop input or Stop (`!S` or `!S1111`) command, motion will decelerate at the preset `AD`/`ADA` value, command execution will be paused, and all commands following the command currently being executed will remain in the command buffer.

Resuming program execution (*only after motion is stopped*):

- Whether stopping as a result of a stop input or Stop (`!S` or `!S1111`) command, you can resume program execution by issuing an immediate Continue (`!C`) command or by activating a pause/resume input (a programmable input configured with the `INFNCi-E` or `LIMFNCi-E` command).
- If you are resuming after a stop input or `!S1111` command, the move in progress will **not** be saved.
- If you are resuming after a `!S` command, you will resume the move in progress at the point in which the `!S` command was received by the processor.

`COMEXS2`:   Upon receiving a stop input or Stop command, motion will decelerate at the preset `AD`/`ADA` value, every command in the buffer will be discarded, and program execution will be terminated, but the `INSELP` value is retained. This allows external program selection, via inputs defined with the `INFNCi-B` (or `LIMFNCi-B`) or `INFNCi-iP` (or `LIMFNCi-iP`) commands, to continue.

Example:
```
COMEXS1           ; Save the command buffer upon a stop input or stop command
```

[COS()]   **Cosine**

			Product	**Rev**
Type	Operator (Trigonometric)		6K	5.0
Syntax	COS(r) (see below)			
Units	r = radians or degrees (depending on RADIAN command)			
Range	r = 0.00000 – ±17500			
Default	n/a			
Response	n/a			
See Also	[ATAN], [PI], RADIAN, [SIN], [TAN], VAR			

Use this operator to calculate the cosine of a number given in radians or degrees (see RADIAN command). If "a" and "b" are coordinates of a point on a circle of radius "r", then the angle of measure "θ" can be defined by the equation:

$$\cos \theta = \frac{b}{r} \qquad \text{(see illustration at right)}$$

If a value is given in radians and a conversion is needed to degrees, or vice-versa, use the formula:

360° = 2π radians.

$$\sin \theta = \frac{a}{r}$$

$$\cos \theta = \frac{b}{r}$$

$$\tan \theta = \frac{a}{b}$$

y = Cos x
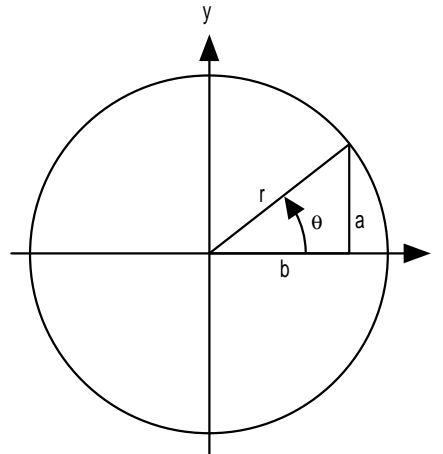
The graph to the right shows the amplitude of **y** on the unit circle for different values of **x**.

2π Radians = 360 Degrees
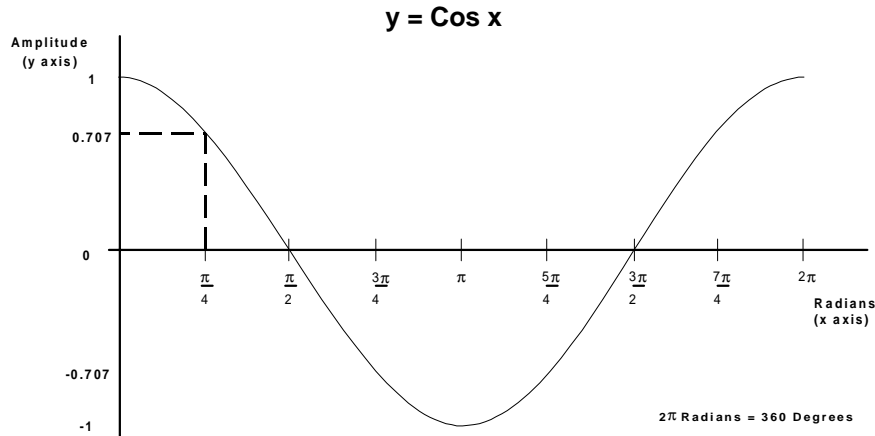
Syntax: VARi=COS(r) where i is the variable number and r is a value in either radians or degrees depending on the RADIAN command. Parentheses (( )) must be placed around the COS operand. **The result will be specified to 5 decimal places.**

Example:
```
VAR1=5 * COS(PI/4)   ; Set variable 1 equal to 5 times the cosine of
                     ; π divided by 4
```

D   Distance

Type	`Motion`	**Product**	**Rev**
Syntax	`<!><@><a>D<r>,<r>,<r>,<r>,<r>,<r>,<r>,<r>`	6K	5.0
Units	`r = distance units (scalable by SCLD)`		
Range	`±999,999,999.99999`		
Default	`4000`		
Response	`D:  *D+4000,+4000,+4000,+4000 ...`		
	`1D:  *1D+4000`		
See Also	`[ D ], GO, MA, MC, PSET, SCLD, TSTAT`		

The Distance (`D`) command defines either the number of units the motor will move or the absolute position it will seek after a `GO` command. In the incremental mode (`MAØ`), the distance value represents the total number of units you wish the motor to move. In the absolute mode (`MA1`) the distance value represents the absolute position the motor will end up at; the actual distance traveled will vary depending on the absolute position of the motor before the move is initiated.

In the incremental mode (`MAØ`), you can specify a negative distance by placing a dash or hyphen (-) in front of the distance value (e.g., `D-10000`). Otherwise, the direction is considered positive. You can change direction without changing the distance value by using the `+`, `-`, or `~` operators (e.g. `D+,+,+`, or `D-,-,-`, or `D~,~,~`); the tilde (`~`) is a means of toggling the direction.

The distance remains set until you change it with a subsequent distance command. Distances outside the valid range are flagged as an error, returning the message `*INVALID DATA-FIELD x`, where `x` is the field number.

UNITS OF MEASURE and **SCALING**: refer to page 16.

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

Example:
```
DEL proga        ; Delete program called proga
DEF proga        ; Begin definition of program called proga
MA0000           ; Incremental index mode for all axes
MC0000           ; Preset index mode for all axes
A10,12,1,2       ; Set the acceleration to 10, 12, 1, and 2 units/sec/sec for
                 ; axes 1, 2, 3 and 4, respectively
AD1,1,1,2        ; Set the deceleration to 1, 1, 1, and 2 units/sec/sec for
                 ; axes 1, 2, 3 and 4, respectively
V1,1,1,2         ; Set the velocity to 1, 1, 1, and 2 units/sec for
                 ; axes 1, 2, 3 and 4, respectively
D100000,1000,10,100   ; Set the distance to 100000, 1000, 10, and 100 units
                 ; for axes 1, 2, 3 and 4, respectively
GO1100           ; Initiate motion on axes 1 and 2, 3 and 4 do not move
END              ; End definition of program called proga
```

[D]   Distance Assignment

			Product	Rev
Type	Assignment or Comparison		**6K**	**5.0**
Syntax	See below			
Units	distance units (scalable by SCLD)			
Range	±999,999,999.99999			
Default	n/a			
Response	n/a			
See Also	D, GO, MA, MC, PSET, SCLD			

The distance assignment (D) command is used to compare the programmed distance value to another value or variable, or to assign the current programmed distance to a variable.

Syntax: VARn=aD where n is the variable number, and a is the axis number, or [D] can be used in an expression such as IF(1D<25ØØØ). When assigning the distance value to a variable, an axis specifier must always precede the D command (e.g., VAR1=1D) or it will default to axis 1. When making a comparison to the programmed distance, an axis specifier must also be used (e.g., IF(1D<2ØØØØ)). The D value used in any comparison, or in any assignment statement is the programmed D value. If the actual position information is required, refer to the PC command for steppers, or the PE or ANI commands for servos.

> **UNITS OF MEASURE** and **SCALING**: refer to page 16.

Example:
```
IF(2D<25000)      ; If the programmed distance on axis 2 is less than 25000 units,
                  ; then do the statements between the IF and NIF
VAR1=2D*2         ; Variable 1 = programmed distance of axis 2 times 2
D,(VAR1)          ; Set the distance on axis 2 to the value of variable 1
NIF               ; End the IF statement
```

[DAC]   Value of DAC Output

			Product	Rev
Type	Assignment or Comparison		**6K**	**5.0**
Syntax	See below			
Units	Volts			
Range	–10.000 to +10.000			
Default	n/a			
Response	n/a			
See Also	DACLIM, SOFFS, TDAC			

Use the DAC command to compare the value of the DAC (commanded analog control signal output) to another value or variable, or to assign the value of the DAC to a variable.

Syntax: VARn=aDAC where "n" is the variable number, and "a" is the axis number, or DAC can be used in an expression such as IF(1DAC<6). An axis specifier must precede the DAC command, or it will default to axis 1 (e.g., VAR1=1DAC, IF(1DAC<2), etc.).

Example:
```
VAR6=2DAC         ; Set variable #6 equal to the DAC voltage output to axis #2
IF(2DAC>5.0)      ; If the DAC voltage to axis #2 is > 5V, do the IF statement.
TDAC              ; Transfer the current DAC values
NIF               ; End IF statement
```

DACLIM   Digital-to-Analog Converter (DAC) Limit

		Product	Rev
Type	Servo; Controller Setup	6K	5.0
Syntax	<!><@><a>DACLIM<r>,<r>,<r>,<r>,<r>,<r>,<r>,<r>		
Units	r = volts		
Range	0.000 to 10.000		
Default	10.000		
Response	DACLIM: *DACLIM10.000,10.000,10.000,10.000 ...		
	1DACLIM: *1DACLIM10.00		
See Also	[DAC], SOFFS, TDAC		

This command sets the maximum absolute value the commanded analog control signal output can achieve. For example, setting the DAC limit to 8.000V (DACLIM8.ØØØ) will clamp the DAC output range from -8.000 to +8.000. Use the TDAC command to verify the voltage being commanded at the servo controller's analog output.

Example:
```
DACLIM7.000,9.000  ; Axis #1 DAC output is limited to -7.000 to +7.000 volts;
                   ; Axis #2 DAC output is limited to -9.000 to +9.000 volts
```

[DAT]   Data Assignment

		Product	Rev
Type	Data Storage	6K	5.0
Syntax	DATi		
Units	i = data program #		
Range	1-50		
Default	n/a		
Response	n/a		
See Also	DATA, [DATP], DATPTR, DATRST, DATTCH		

The Data Assignment (DAT) command recalls data from the data program (DATP). The data is loaded into a command field, or into a variable (VAR). As the data is loaded, the internal data pointer to the DATP data increments and points to the next datum for the next DAT command.

Syntax: VARn=DATi where "n" is the variable number, and "i" is the data program number, or DAT can be used as a command argument such as A(DAT1),5,4,1Ø

If the data is to be loaded into a command field, the DAT command must be placed within parentheses (e.g., AD(DAT2),3,4,5). If the data is loaded into a variable, parentheses are not required. (e.g., VAR1=DAT2). Rule of Thumb for command value substitutions: If the command syntax shows that the command field requires a real number (denoted by <r>) or and integer value (denoted by <i>), you can use the DAT substitution (e.g., HOMV2,1,(DAT1)).

The DAT command cannot be used in an expression, such as IF(DAT2 < 5) or VAR1=1 + DAT3.

Example: Refer to the Reset Data Pointer (DATRST) command example.

DATA   Data Statement

		Product	Rev
Type	Data Storage	6K	5.0
Syntax	<!>DATA=<r>,<r>,<r>,<r>		
Units	r = data value		
Range	±999,999,999.99999999		
Default	n/a		
Response	n/a		
See Also	[DAT], [DATP], DATPTR, DATRST, DATTCH, MEMORY		

The Data Statement (DATA) command is used only in the data programs (DATP) to identify the data statements. The DATA command is followed by an equal sign (=), and a maximum of four data values. The maximum number of data statements is limited only by the amount of memory available.

Example: Refer to the Reset Data Pointer (DATRST) command example.

[DATP]   Data Program

Type	Data Storage		**Product**	**Rev**
Syntax	DATPi		6K	5.0
Units	i = data program #			
Range	1-50			
Default	n/a			
Response	n/a			
See Also	[DAT], DATA, DATPTR, DATRST, DATSIZ, DATTCH, MEMORY			

DATP is not a command, but is the name of the program that is the default for storing data. Fifty such data programs can be created, DATP1 - DATP5Ø. The program is defined with the DEF command, just as any other program would be, but only the DATA and END commands are allowed within the program definition. DATPi will contain the array of data to be recalled by the DATi command. Upon completion of the definition, the internal data pointer is pointing to the first datum in the data program.

Example:
```
DEF DATP5        ; Define data program 5
DATA=1,2,3,4     ; Enter data
DATA=5.62,6.52,7.12,8.47   ; Enter data
END              ; End program definition
A(DAT5)          ; Load data from data program 5 and store in axis 1 acceleration.
                 ; Axis 1 acceleration = 1
V(DAT5)          ; Load data from data program 5 and store in axis 1 velocity.
                 ; Axis 1 velocity = 2
D(DAT5)          ; Load data from data program 5 and store in axis 1 distance.
                 ; Axis 1 distance = 3
A,(DAT5)         ; Load data from data program 5 and store in axis 2 acceleration.
                 ; Axis 2 acceleration = 4
A,,(DAT5)        ; Load data from data program 5 and store in axis 3 acceleration.
                 ; Axis 3 acceleration = 5.62
```

DATPTR   Set Data Pointer

Type	Data Storage		**Product**	**Rev**
Syntax	<!>DATPTRi,i,i		6K	5.0
Units	n/a			
Range	1st i = program # 1 to 50			
	2nd i = data element # 1 to 6500			
	3rd i = increment setting of 1 to 100			
Default	1,1,1			
Response	n/a			
See Also	[DAT], DATA, [DATP], DATSIZ, DATTCH, [DPTR], TDPTR			

The Set Data Pointer (DATPTR) command moves the internal data pointer to a specific data element in the specified data program (DATPi). This command also establishes the number of data elements by which the pointer increments after writing each data element from a DATTCH command, or after recalling a data element with the DAT command.

The data program selected with the first integer in the DATPTR command becomes the active data program. Subsequent DATTCH, TDPTR, and DPTR commands will reference the active data program. You can use the TDPTR command to ascertain the current active data program, as well as the current location of the data pointer and the increment setting (see TDPTR command description for details).

The DPTR command can be used to compare the current pointer location (the number of the data element to which the data pointer is pointing) against another value or variable, or to assign the pointer location number to a variable.

As an example, suppose data program #1 (DATP1) is configured to hold 15 data elements (DATSIZ1,15), the data pointer is configured to start at the first data element and increment 1 data element after every DATTCH value is stored (DATPTR1,1,1), and the values of numeric variables #1 through #4 are already assigned (VAR1=2, VAR2=4, VAR3=8, VAR4=64). If you then enter the DATTCH1,2,3,4 command, the values of VAR1 through VAR4 will be assigned respectively to the first four data elements in the data program and the pointer will stop at data element #5. The response to the TPROG DATP1 command would be

as depicted below (the text is highlighted to illustrate the location of the data pointer after the `DATTCH1,2,3,4` command is executed). The response to the `TDPTR` command would be `*TDPTR1,5,1`.

```
*DATA=2.0,4.0,8.0,64.0
*DATA=0.0,0.0,0.0,0.0
*DATA=0.0,0.0,0.0,0.0
*DATA=0.0,0.0,0.0
```

Once you have stored (taught) the variables to the data program, you can use the `DATPTR` command to point to the data elements and then use the `DAT` data assignment command to read the stored variables to your motion program.

During the process of writing data (`DATTCH`) or recalling data (`DAT`), if the pointer reaches the last data element in the program, it automatically wraps around to the first datum in the program and a warning message is displayed (`*WARNING: POINTER HAS WRAPPED AROUND TO DATA POINT 1`). This warning will not interrupt command execution.

Example: (See Also: `DATSIZ` command)
```
DEL DATP5         ; Delete data program #5 (DATP5)
DEF DATP5         ; Define data program #5 (DATP5)
DATA=1,2,3,4      ; Enter data
DATA=5.62,6.52,7.12,8.47   ; Enter data
END               ; End program definition
A(DAT5)           ; Load data from DATP5 and store in axis 1 acceleration.
                  ; Axis 1 acceleration = 1
V(DAT5)           ; Load data from DATP5 and store in axis 1 velocity.
                  ; Axis 1 velocity = 2
D(DAT5)           ; Load data from DATP5 and store in axis 1 distance.
                  ; Axis 1 distance = 3
DATPTR5,1,1       ; Set the data pointer to datum 1 in DATP5; increment the
                  ; pointer by one after each DAT command
A,(DAT5)          ; Load data from DATP5 and store in axis 2 acceleration.
                  ; Axis 2 acceleration = 1
A,,(DAT5)         ; Load data from DATP5 and store in axis 3 acceleration.
                  ; Axis 3 acceleration = 2
```

DATRST   Reset Data Pointer

Type	Data Storage	**Product**	**Rev**
Syntax	<!>DATRST<i>,<i>	6K	5.0
Units	n/a		
Range	1st i = program # 1 to 50, 2nd i = data element # 1 to 6500		
Default	n/a		
Response	n/a		
See Also	[DAT], DATA, [DATP]		

The Reset Data Pointer (`DATRST`) command sets the internal data pointer to a specific data element in a data program (`DATP<i>`). As data is recalled from a data program with the `DAT` command, the pointer automatically increments to the next data element. If the pointer reaches the end of the program, it automatically wraps around to the first data element in the program. `DATRST` allows the pointer to be set to any location within the data program (`DATP`).

Example：
```
DEF DATP5         ; Define data program 5
DATA=1,2,3,4      ; Enter data
DATA=5.62,6.52,7.12,8.47   ; Enter data
END               ; End program definition
A(DAT5)           ; Load data from data program 5 and store in axis 1 acceleration.
                  ; Axis 1 acceleration = 1
V(DAT5)           ; Load data from data program 5 and store in axis 1 velocity.
                  ; Axis 1 velocity = 2
D(DAT5)           ; Load data from data program 5 and store in axis 1 distance.
                  ; Axis 1 distance = 3
DATRST5,1         ; Set the data pointer to datum 1 in data program 5
A,(DAT5)          ; Load data from data program 5 and store in axis 2 acceleration.
                  ; Axis 2 acceleration = 1
A,,(DAT5)         ; Load data from data program 5 and store in axis 3 acceleration.
                  ; Axis 3 acceleration = 2
```

DATSIZ   Data Program Size

Type	Data Storage		**Product**	**Rev**
Syntax	<!>DATSIZi<,i>		6K	5.0
Units	n/a			
Range	1st i = program # 0 - 50 (0 = disable)			
	2nd i = data element # 1 - 6500			
Default	0,1			
Response	n/a			
See Also	[DAT], DATPTR, [DATP], DATTCH			

The Data Program Size (DATSIZ) command creates a new data program (DATP) and establishes the number of data elements the data program contains.

The DATSIZ command syntax is DATSIZi<,i>. The first integer (i) represents the number of the data program (1 - 50). You can create up to 50 separate data programs. The data program is automatically given a specific program name (DATPi). If the program number Ø is selected, then the DATTCH command is disabled. Before creating a new data program, be sure to delete the existing data program that has the same name. For example, if you wish to create data program #5 with the DATSIZ5,1,144 command and DATP5 already exists, first delete DATP5 with the DEL DATP5 command and then issue the DATSIZ5,1,144 command.

The second integer represents the total number of data elements (up to 6,500) you want in the data program. Upon issuing the DATSIZ command, the data program is created with all the data elements initialized with a value of zero. (The DATSIZ command is equivalent to creating a DATP program and filling it with DATA=Ø.Ø,Ø.Ø,Ø.Ø,Ø.Ø commands up to the size indicated in the second integer.)

Each data statement, which contains four data elements, uses 43 bytes of memory. This amount of memory is subtracted from the memory allocated for user programs (see MEMORY command). Use the TDIR command to determine the amount of remaining memory for user program storage.

The data program has a tabular structure, where the data elements are stored 4 to a line. Each line of data elements is called a *data statement*. Each element is numbered in sequential order from left to right (1 - 4) and top to bottom (1 - 4, 5 - 8, 9 - 12, etc.). You can use the TPROG DATPi command ("i" represents the number of the data program) to display all the data elements of the data program. For example, if you issue the DATSIZ1,13 command, data program #1 (called DATP1) is created with 13 data elements initialized to zero. The response to the TPROG DATP1 command is depicted below. Each line (*data statement*) begins with DATA=, and each data element is separated with a comma.

```
*DATA=Ø.Ø,Ø.Ø,Ø.Ø,Ø.Ø
*DATA=Ø.Ø,Ø.Ø,Ø.Ø,Ø.Ø
*DATA=Ø.Ø,Ø.Ø,Ø.Ø,Ø.Ø
*DATA=Ø.Ø
```

The DATSIZ, DATTCH, and DAT commands will typically be used as a teach mode in this manner:

1. Issue the DATSIZ command to create (or recall) the data program.

2. Store variable data (e.g., position, acceleration, velocity, etc.) to numeric variables (VAR).

3. Use DATTCH commands to store the data from the numeric variables into the data program. You can use the data pointer (DATPTR) command to select any data element in the data program, and to determine the number by which the pointer increments after each value from the DATTCH command is stored. *NOTE: If the DATTCH command is issued without having issued the DATSIZ command, an error will result.*

4. Use the DAT commands to read the stored data from the data program into the variable parameters of your motion program. You can use the DATPTR command to select any data element in the data program, and to determine the number by which the pointer increments after each DAT command.

Any use of the DATTCH and DAT commands will reference the current active data program (DATP) specified by the first integer of the last DATSIZ or DATPTR command. If you want to use the DATSIZ command to recall a data program, **and not create one**, specify only the first integer and not the second integer. For example, DATSIZ7 recalls data program #7 (DATP7) as the active data program.

Example (for 4 axes):

```
DEL DATP5          ; Delete existing data program #5 (DATP5)
DATSIZ5,200        ; Create data program #5 (DATP5) with 200 data elements
DEF TEACH          ; Begin definition of program called TEACH
COMEXC0            ; Disable continuous command execution mode
MA1111             ; Enable the absolute positioning mode for all axes
HOM1111            ; Home all axes (absolute position counter set to zero after homing)
DATPTR5,1,1        ; Set data pointer to data element #1 in DATP5, and increment the
                   ; pointer by one element after every DATTCH value or DAT command
REPEAT             ; Set up a loop for teaching the positions
JOY1111            ; Enable joystick mode on all axes so that you can start moving the
                   ; axes into position with the joystick. Command processing stops
                   ; here until you activate trigger input TRG-A1 (IN.2) to disable the
                   ; joystick mode and execute the rest of the commands in the
                   ; repeat/until loop (assign the motor positions to the variables and
                   ; then store the positions from the variables to the data program).
VAR1=1PM           ; Store the current position of axis #1 in variable #1
VAR2=2PM           ; Store the current position of axis #2 in variable #2
VAR3=3PM           ; Store the current position of axis #3 in variable #3
VAR4=4PM           ; Store the current position of axis #4 in variable #4
DATTCH1,2,3,4      ; Store variables #1 - #4 into consecutive data elements
WAIT(IN.2=b0)      ; Wait for the "joystick release" input (TRG-A1) to be de-activated
UNTIL(DPTR=1)      ; Repeat loop until the data pointer wraps around to data element #1
HOM1111            ; Home all axes (absolute position counter set to zero after homing)
DATPTR5,1,1        ; Set data pointer to data element #1, read one data element at a time
REPEAT             ; Set up a repeat/until loop to read all data elements
D(DAT5),(DAT5),(DAT5),(DAT5)     ; Read position data from the data program to the
                   ; distance command
GO1111             ; Make the move to the positions that were taught
T.2                ; Wait 0.2 seconds
UNTIL(DPTR=1)      ; Repeat loop until the data pointer wraps around to data element #1
END                ; End definition of program called TEACH
```

DATTCH   Data Teach

		Product	Rev
Type	Data Storage	6K	5.0
Syntax	<!>DATTCHi<,i,i,i>		
Units	i = number of a numeric variable		
Range	i = 1 – maximum number of numeric variables		
Default	n/a		
Response	n/a		
See Also	[DAT], [DATP],DATPTR, DATSIZ, DATTCH, VAR		

The Data Teach (DATTCH) command stores the values from the specified numeric variables (VAR) into the **currently active data program** (i.e., the data program specified with the last DATSIZ or DATPTR command). The value that is in the specified variable at the time the DATTCH command is executed is the value that is stored in the data program.

If the DATTCH command is issued without having first issued the DATSIZ command, an error will result. If a zero is entered in the first integer of the DATSIZ command (e.g., DATSIZØ), the DATTCH command is disabled.

As indicated by the number of integers in the syntax, the maximum number of variables that can be stored in the data program per DATTCH command is 4. The variables are stored in the data program, starting at the current location of the data pointer. The data pointer's position can be moved to any data element in any data program by use of the DATPTR command. After each successive DATTCH value is stored, the data pointer will increment by the number specified in the third integer of the DATPTR command. Any data element in the data program can be edited by setting the data pointer to that element and then issuing the DATTCH command.

As an example, suppose data program #1 (DATP1) is configured to hold 15 data elements (DATSIZ1,15), the data pointer is configured to start at the first data element and increment 1 data element after every DATTCH value is stored (DATPTR1,1,1), and the values of numeric variables #1 through #4 are already assigned (VAR1=2, VAR2=4, VAR3=8, VAR4=64). If you then enter the DATTCH1,2,3,4 command, the values of VAR1 through VAR4 will be assigned respectively to the first four data elements in the data

program and the pointer will stop at data element #5. The response to the TPROG DATP1 command would be as follows (the text is highlighted to illustrate the location of the data pointer after the DATTCH1,2,3,4 command is executed).

```
*DATA=2.Ø,4.Ø,8.Ø,64.Ø
*DATA=Ø.Ø,Ø.Ø,Ø.Ø,Ø.Ø
*DATA=Ø.Ø,Ø.Ø,Ø.Ø,Ø.Ø
*DATA=Ø.Ø,Ø.Ø,Ø.Ø
```

Example: Refer to the DATSIZ command**.**

DCLEAR   **Clear Display**

		Product	**Rev**
Type	Display (RP240) Interface	6K	5.0
Syntax	<!>DCLEARi		
Units	n/a		
Range	i = 0 (clear all lines), 1 (clear line 1), or 2 (clear line 2)		
Default	n/a		
Response	n/a		
See Also	DLED, DPASS, DPCUR, DSTP, DVAR, DVARB, DVARI, DWRITE		

The Clear Display (DCLEAR) command clears lines (as specified with i) of the RP240 display. After clearing a line, the cursor will be reset to the beginning of that line (or to the beginning of line 1 if all lines are cleared).

DEF   **Begin Program/Subroutine/Path Definition**

		Product	**Rev**
Type	Program or Subroutine Definition	6K	5.0
Syntax	<!>DEF<t>		
Units	t = alpha text string (name of a program)		
Range	text string of 6 characters or less		
Default	n/a		
Response	n/a		
See Also	$, DEL, END, ERASE, GOBUF, GOSUB, GOTO, MEMORY, PCOMP, PLCP, PRUN, RUN, [SS], TDIR, TMEM, TPROG, TSS, TSTAT		

The Define a Program/Subroutine (DEF) command is the beginning of a program, path contour, or subroutine definition. The syntax for the command is DEF followed by 6 or fewer alpha-numeric characters. The first character may not be a number. Refer to the MEMORY command description for information on program size restriction and total number of programs possible per product.

All programs are stored in a binary fashion within the 6K Series products. A program transferred back out (TPROG) after it has been defined (DEF), may not look identical to the program defined. However, the program is functionally identical.

NOTE
When defining a program and the memory limitation is exceeded, an error message will be generated, and bit 11 of the system status register will be set (SS or TSS). The program will be stored up to the point where the memory limitation was exceeded.

There is no actual difference in the definition of, or execution of a program versus the definition, or execution of a subroutine. Both a program and a subroutine are defined as the set of commands between a DEF<t> and an END command. If an invalid program/subroutine name is entered, an error message will be generated. An invalid program/subroutine name is any name that is also a current command (An example of an invalid name would be DEFhomx, because it is impossible for the operating system to distinguish the homx subroutine call from the HOMx111 go home command.). A subroutine/program definition cannot be assigned the name "CLR" and cannot contain any of the following characters:
!, –, #, $, %, ^, &, *, (,), +, –, _, =, {, }, \, |, ", :, ;, ', <, >, ,, ., ?, /.

The RUN command can be used to start executing a program/subroutine. The program name by itself can also be used to start executing a program/subroutine. A compiled profile (contour or compiled motion profile) must be compiled with the PCOMP command before it can be executed with the PRUN command; and a compiled PLC (PLCP) program must be compiled with PCOMP before is can be executed with the SCANP or PRUN command.

The GOTO and GOSUB commands can be used within a program/subroutine to go to another program/subroutine.

NOTE: Program, compiled profile, or subroutine names must be deleted (DEL) before they can be redefined.

Example:
```
DEL pick            ; Delete program named pick
DEF pick            ; Begin definition of program named pick
GO1100              ; Initiate motion on axes 1 and 2, not on axes 3 and 4
END                 ; End program definition
RUN pick            ; Execute program pick
```

DEL   Delete a Program/Subroutine/Path

Type	Program or Subroutine Definition	**Product**	**Rev**
Syntax	<!>DEL<t>	6K	5.0
Units	t = alpha text string (name of a program)		
Range	text string of 6 characters or less		
Default	n/a		
Response	n/a		
See Also	$, DEF, END, ERASE, GOSUB, GOTO, RUN		

The Delete a Program/Subroutine (DEL) command removes a program, path contour, or subroutine definition. The syntax for the command is DEL followed by 6 or fewer alpha-numeric characters. To delete all programs refer to the ERASE command. The DEL command can be placed inside a program (e.g., to delete a DATP program).

To edit an existing program, you must first delete it. The DEL command will not delete a label ($).

Example:
```
DEF pick            ; Begin definition of program named pick
GO1100              ; Initiate motion on axes 1 and 2
END                 ; End program definition
RUN pick            ; Execute program pick
DEL pick            ; Deletes program named pick
```

DJOG   Enable RP240 Jog Mode

Type	Display (RP240) Interface	**Product**	**Rev**
Syntax	<!>DJOG<b>	6K	5.0
Units	b = 0 or 1		
Range	0 = disable, 1 = enable		
Default	0		
Response	DJOG: *DJOG0		
See Also	JOG, JOGA, JOGAA, JOGAD, JOGADA, JOGVH, JOGVL		

The DJOG command allows you to branch into the RP240 front panel jog mode from within your user-defined program, adjust the position of the axes, and then return to program execution.

The DJOG1 command enables the RP240 jog mode on all axes. Once the RP240 jog mode is enabled, you can use the RP240 arrow keys to jog individual axes. Unlike the JOG command, command processing is suspended after the DJOG1 command is issued. Jogging acceleration and deceleration are performed with the parameters set with the Jog Acceleration (JOGA) and Jog Deceleration (JOGAD) commands. Jogging velocities are set with the Jog Velocity High (JOGVH) and the Jog Velocity Low (JOGVL) commands. Once in the RP240 Jog Mode, you can switch between low and high jog velocities for any axis, and you can also modify the two jog velocities using the RP240's **EDIT** key.

To disable the RP240 jog mode, press the **MENU RECALL** key or issue the immediate !DJOGØ command. Upon exiting the RP240 jog mode, the RP240's display is cleared.

To have the jog mode continually enabled during program execution, you must use jog inputs and the JOG command.

[DKEY]   Value of RP240 Key

		Product	**Rev**
Type	Display (RP240) Interface; Assignment or Comparison	6K	5.0
Syntax	See below		
Units	n/a		
Range	n/a		
Default	n/a		
Response	n/a		
See Also	DCLEAR, DPCUR, [DREAD], DREADF, DREADI, DVAR, DWRITE		

The DKEY operator allows you to read the current state of the RP240 key-pad and use it in comparison commands (e.g., IF, WHILE, etc.) or variable assignments. **NOTE**: If two or more keys are pressed simultaneously, DKEY will report -1.

Syntax: VARn=DKEY where "n" is the variable number,
              or DKEY can be used in an expression such as IF(DKEY=-1)

The value reported by the DKEY command is defined by the following table:

Value of DKEY	**Key currently active**
-1	None or multiple keys
0	0
1	1
2	2
3	3
4	4
5	5
6	6
7	7
8	8
9	9
10	.
11	+/-
12	C/E
13	ENTER
14	Menu Recall
15	STOP
16	PAUSE
17	CONTINUE
21	F1
22	F2
23	F3
24	F4
25	F5
26	F6

DLED   Turn RP240 Display LEDs On/Off

		Product	**Rev**
Type	Display (RP240) Interface	6K	5.0
Syntax	`<!>DLED<b><b><b><b><b><b><b><b>`		
Units	n/a		
Range	b = 0 (off) or 1 (on)		
Default	n/a		
Response	DLED: *DLED1101_0001		
See Also	DCLEAR, DPASS, DPCUR, DSTP, DVAR, DVARB, DVARI, DWRITE		

The `DLED` command controls the state of the 8 programmable LEDs on the RP240. *It is legal to substitute a binary variable (`VARB`) for the `DLED` command.*

Example:
```
DLED11XXXX01     ; Turn on LEDs 1, 2, and 8; turn off LED 7; leave LEDs 3,4,5,
                 ; and 6 unchanged
VARB1=b10101010  ; Set bits 1, 3, 5 & 7 low, and bits 2, 4, 6, & 8 high
DLED(VARB1)      ; Turn on LEDs 1, 3, 5 & 7; turn off LEDs 2, 4, 6, & 8
```

DPASS   Change RP240 Password

		Product	**Rev**
Type	Display (RP240) Interface	6K	5.0
Syntax	`<!>DPASS<i>`		
Units	i = integer of up to 9 characters		
Range	1 – 9999		
Default	For the 6K: DPASS6850		
Response	DPASS: *DPASS6850		
See Also	DCLEAR, DLED, DPCUR, DSTP, DVAR, DVARB, DVARI, DWRITE		

The `DPASS` command changes the RP240 password. If the default password is not changed by the user, then there will be no password protection.

Example:
```
DPASS2001        ; New password = 2001
```

DPCUR   Position Cursor

		Product	**Rev**
Type	Display (RP240) Interface	6K	5.0
Syntax	`<!>DPCURi,i`		
Units	1st i = line number, 2nd i = column		
Range	line number = 1 or 2, column = 0 – 39		
Default	n/a		
Response	n/a		
See Also	DCLEAR, DLED, DPASS, DREADI, DSTP, DVAR, DVARI, DVARB, DWRITE		

The Position Cursor (`DPCUR`) command changes the location of the cursor on the RP240 display. The RP240 lines are numbered from top to bottom, 1 to 2. The columns are numbered left to right, 0 to 39.
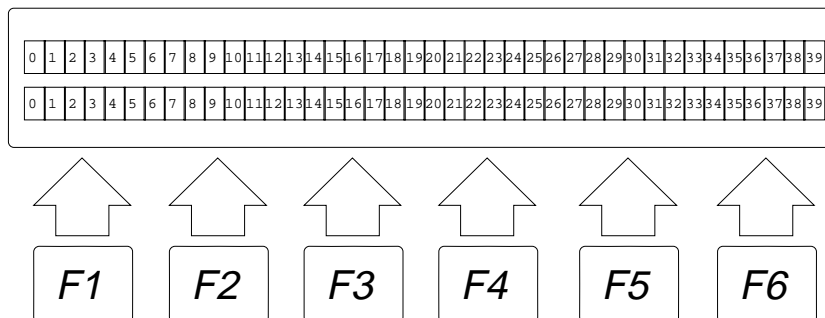
Example:
```
DPCUR2,15        ; Position cursor on line 2, column 15
```

[DPTR]   Data Pointer Location

Type	Data Storage; 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	[DAT], DATA, [DATP], DATPTR, DATSIZ, TDPTR			

The DPTR command can be used to compare the current pointer location (the number of the data element to which the data pointer is pointing) against another value or numeric variable, or to assign the pointer location number to a variable. The current data pointer location is referenced to the current active data program specified in the first integer of the last DATSIZ or DATPTR command.

Syntax: VARn=DPTR where "n" is the variable number,
          or DPTR can be used in an expression such as IF(DPTR=1)

Example：
```
DATSIZ4,200     ; Create data program called DATP4 with 200 data elements
DATPTR4,20,2    ; Set the data pointer to data element #20 in DATP4 and set the
                ; increment to 2 (DATP4 becomes the current active data program)
VAR1=DPTR       ; Assign the number of the pointer location in DATP4 to numeric
                ; variable #1
VAR1            ; Response is *VAR1=20. Indicates that the data pointer is
                ; pointing to data element #20.
```

[DREAD]   Read RP240 Data

Type	Display (RP240) Interface		**Product**	**Rev**
Syntax	See below		6K	5.0
Units	n/a			
Range	n/a			
Default	n/a			
Response	n/a			
See Also	[DREADF], DREADI, DVAR, DWRITE, [SS], TSS, VAR			

The Read RP240 Data (DREAD) command allows you to store numeric data entered in from the RP240's keypad into a variable. As the user presses RP240 numeric keys, the data will be displayed on the RP240 starting at the location equal to the current cursor location + 1 (for a sign bit):

    VAR1=DREAD   Wait for RP240 numeric entry (terminated with the **ENTER** key), then set VAR1 equal to that value.

Additionally the DREAD command can be used as a variable assignment within another command that is expecting numeric data (<u>Rule of Thumb</u>: If the command syntax shows that the command field requires a real number (denoted by <r>) or and integer value (denoted by <i>), you can use the DREAD substitution.):

    A(DREAD),5.0   Wait for RP240 numeric entry (terminated with the **ENTER** key), then set axis #1 acceleration to that value and set axis #2 acceleration to 5.0.

The DREAD command cannot be used in an expression such as VAR5=4+DREAD or IF(DREAD=1).

[DREADF]   Read RP240 Function Key

Type	Display (RP240) Interface	**Product**	**Rev**
Syntax	See below	6K	5.0
Units	n/a		
Range	n/a		
Default	n/a		
Response	n/a		
See Also	[DREAD], DREADI, DVAR, DWRITE, [SS], TSS, VAR, VARI		

The Read RP240 Function Key (DREADF) command allows you to store numeric data entered in from a RP240 function key into a variable. Function key 1 (**F1**) = 1, **F2** = 2, etc., and **MENU RECALL** (**F0**) = 0.

<u>Rule of Thumb for command value substitutions:</u> If the command syntax shows that the command field requires a real number (denoted by <r>) or and integer value (denoted by <i>), you can use the DREADF substitution (e.g., V2,(DREADF)).

Example:
```
VAR1=DREADF       ; Wait for RP240 function key entry, then set VAR1 equal to that
                  ; value
IF(VAR1=5)        ; If function key 5 was hit then ...
GOx1              ; Start motion on axis #2
NIF               ; End if statement
```

DREADI   RP240 Data Read Immediate Mode

Type	Display (RP240) Interface	**Product**	**Rev**
Syntax	<!>DREADI<b>	6K	5.0
Units	n/a		
Range	1 (enable) or 0 (disable)		
Default	0		
Response	DREADI: *DREADI0		
See Also	DPCUR, [DREAD], [DREADF], VAR, VARI		

The DREADI1 command allows continual numeric or function key data entry from the RP240 (when used in conjunction with the DREAD and/or DREADF commands). In this immediate mode, program execution is not paused (waiting for data entry) when a DREAD or DREADF command is encountered.

NOTES

- While in the Data Read Immediate Mode (DREADI1), data is read into VAR and VARI variables only (e.g., A(DREAD) or V(DREAD) substitutions are not valid).

- This feature is not designed to be used in conjunction with the RP240's standard menus (see *Programmer's Guide* for menu structure); the **RUN** and **JOG** menus will disable the DREADI mode.

- After the RP240's ENTER key is pressed (to enter numeric data), the value is displayed on the RP240 display at the 1,30 location (showing 10 significant digits).

- Do not assign the same variable to read numeric data **and** function key data—pick only one.

Simple Numeric Data Entry (example):
```
VAR1=25000       ; Initialize variable #1
DCLEAR0          ; Clear entire RP240 display
DWRITE"ENTER VALUE > " ; Send message to RP240 display starting at location 1,0
DREADI1          ; Enable RP240 data read immediate mode
VAR1=DREAD       ; Set variable #1 (VAR1) to receive data entered on the RP240.
                 ; Current VAR1 data will be displayed at cursor location 1,30
                 ; (fixed). New data will be displayed at current cursor location
                 ; as defined by the previous DCLEAR, DWRITE and DPCUR commands—
                 ; this is the home cursor location for subsequent data entries.
L77              ; Start loop of 77 repetitions
D(VAR1)          ; Set distance equal to the current (last entered) RP240 data
GO1              ; Initiate move on axis one
LN               ; End loop
DREADI0          ; Exit RP240 data read immediate mode
```

```
; As the loop is running, the user may enter in a new distance value
; (which must be terminated with the ENTER key) via the RP240 numeric keypad.
; The numeric keystrokes cause the digits to be displayed on the RP240
; starting at the home cursor location (see VAR1=DREAD description in the
; example above). When the ENTER key is pressed, the variable is updated;
; the most significant 10 digits (total, including sign & decimal point
; if appropriate) of this variable are displayed at cursor location 1,30;
; and then the data entry field (starting at home) is cleared.
; The 6K controller is ready to accept new data.
```

Numeric Data & Function Key Entry (example):
```
VAR1=25000       ; Initialize variable #1
VAR2=1           ; Initialize variable #2
DCLEAR0          ; Clear the RP240 display
DPCUR2,0         ; Place RP240 cursor on line 2, column 0 (bottom left corner of
                 ; display)
DWRITE" SLOW FAST"    ; Send message to RP240 display starting at location 2,0
DPCUR1,0         ; Place RP240 cursor on line 1, column 0 (top left corner)
DWRITE"ENTER VALUE > "  ; Send message to RP240 display starting at location 1,0
DREADI1          ; Enable RP240 data read immediate mode
VAR1=DREAD       ; Set variable #1 (VAR1) to receive numeric data entered on the
                 ; RP240's keypad
VAR2=DREADF      ; Set VAR2 to receive RP240 function key input
L                ; Begin loop
IF(VAR2=1)       ; If function key 1 was last pressed, do the IF statement
                 ; (slow velocity)
V3.6             ; Set velocity to 3.6 units per second
NIF              ; End IF statement
IF(VAR2=2)       ; If function key 2 was last pressed, do the IF statement
                 ; (fast velocity)
V6.4             ; Set velocity to 6.4 units per second
NIF              ; End IF statement
D(VAR1)          ; Set distance equal to the current (last entered) RP240 numeric
                 ; data
GO1              ; Initiate the move on axis one
LN               ; End loop
; As the loop is running, the user may enter in a new distance value and/or
; choose between two different preset velocities. The display does not change
; when a function key is pressed.
```

Multiple Numeric Data Entry (example):
```
VAR2=0           ; Initialize variable #2 (VAR2)
VAR3=99          ; Initialize variable #3 (VAR3)
VAR4=10          ; Initialize variable #4 (VAR4)
VAR5=25000       ; Initialize variable #5 (VAR5)
DCLEAR0          ; Clear the entire RP240 display
DPCUR2,0         ; Place RP240 cursor on line 2, column 0 (bottom left corner)
DWRITE" ACCEL VEL DIST"  ; Send message to RP240 display starting at location 2,0
DREADI1          ; Enable RP240 data read immediate mode
VAR2=DREADF      ; VAR2 will capture function key entries (0 - 6)
L                ; Begin loop
IF(VAR2<>0)      ; If a new function key is pressed, do the following code:
DCLEAR1          ; Clear line one of the RP240 display (top line)
IF(VAR2=1)       ; If function key 1 is pressed, do the IF statement
                 ; (input acceleration)
DWRITE"ENTER ACCEL VALUE>" ; Send message to RP240 display starting at location 1,0
VAR3=DREAD       ; Set VAR3 equal to the numeric data entered on the RP240's keypad
NIF              ; End IF statement
IF(VAR2=2)       ; If function key 2 is pressed, do the IF statement (input velocity)
DWRITE"ENTER VEL VALUE>" ; Send message to RP240 display starting at location 1,0
VAR4=DREAD       ; Set VAR4 equal to the numeric data entered on the RP240's keypad
NIF              ; End IF statement
IF(VAR2=3)       ; If function key 3 is pressed, do the IF statement (input distance)
DWRITE"ENTER DIST VALUE>" ; Send message to RP240 display starting at location 1,0
VAR5=DREAD       ; Set VAR5 equal to the numeric data entered on the RP240's keypad
NIF              ; End IF statement
VAR2=0           ; Prohibit repeated execution of this code
VAR2=DREADF      ; Re-enable VAR2 to capture new function key entry
NIF              ; End IF statement
A(VAR3)          ; Set acceleration equal to the numeric value of VAR3
```

```
V(VAR4)          ; Set velocity equal to the numeric value of VAR4
D(VAR5)          ; Set distance equal to the numeric value of VAR5
GO1              ; Initiate the move on axis one
LN               ; End loop
; As the loop is running, the user may select among the three variables he wants
; to enter data into. These three variables correspond with acceleration,
; velocity, and distance. Each time the function key variable changes from 0
; (to 1, 2 or 3), then a new message is displayed and the VARi=DREAD command
; will put the current value of that variable in location 1,30 (upper right hand
; corner of the display). For example, the user can choose VEL (F2) and then
; repeatedly change VAR4 by entering a value on the RP240 numeric keypad and
; pressing the ENTER key. Each time through the loop, the VAR4 data is loaded
; into the V command.
```

DRES   Drive Resolution

		Product	Rev
Type	Drive Configuration	6K	5.0
Syntax	`<!><@><a>DRES<i>,<i>,<i>,<i>,<i>,<i>,<i>,<i>`		
Units	i = steps/rev		
Range	200-1024000	(applicable only to	
Default	4000	stepper axes)	
Response	DRES: *DRES4000,4000,4000,4000 ...		
	1DRES: *1DRES4000		
See Also	DRFEN, DRFLVL, DRIVE, ERES, PULSE, SCALE, TSTAT		

The Drive Resolution (DRES) command is used to match the controller resolution to that of the motor/drive to which it is attached. This command is necessary in order to accurately calculate motor drive accelerations and velocities whether scaling is disabled (SCALEØ), or enabled (SCALE1).

Example:
```
DRES200,10000,25000,25000 ; Set drive res. for axis 1 to 200 steps/rev, axis 2
                          ; to 10000 steps/rev, and axes 3 & 4 to 25000 steps/rev
```

DRFEN   Enable/Disable Checking the Drive Fault Input

		Product	Rev
Type	Drive Configuration	6K	5.0
Syntax	`<!><@><a>DRFEN<b><b><b><b><b><b><b><b>`		
Units	b = enable bit		
Range	b = 1 (check the state of the drive fault input),		
	0 (don't check the state of the drive fault input), or		
	x (don't change)		
Default	0 (disabled)		
Response	DRFEN: *DRFEN0000_0000		
	1DRFEN: *1DRFEN0		
See Also	DRFLVL, DRIVE, [AS], [ASX], [ER], ERROR, TAS, TASX, TER		

Use the DRFEN command to enable or disable checking the state of the drive fault input for each axis. The default condition is that the drive fault input is not checked (DRFEN0); therefore, a drive fault would not be detectable. Even with DRFEN enabled (DRFEN1), the controller will not respond to a drive fault condition until the respective axis is enabled with the DRIVE1 command.

DRFEN1 is required before you can use these functions (remember that the default power-up state is DRFEN0):

- AS, TAS, and TASF (axis status) bit #14 reports if a drive fault occurred.
- ERROR bit #4 enables checking for the occurrence of a drive fault, and when is does, to branch to the ERRORP program.
- ER, TER, and TERF (error status) bit #4 reports if a drive fault occurred (if ERROR bit #4 is enabled).
- An output assigned the "Fault Indicator" function (OUTFNCi-F) will turn on when a drive fault occurs or a user fault input (INFNCi-F or LIMFNCi-F) is activated.

Regardless of the state of the DRFEN command, ASX, TASX, and TASXF (extended axis status) bit #4 will accurately report the hardware state of the drive fault input.

The DRFEN command setting is not saved in the controller's battery backed RAM.

DRFLVL   Drive Fault Level

		Product	**Rev**
Type	Drive Configuration	6K	5.0
Syntax	<!><@><a>DRFLVL<b><b><b><b><b><b><b><b>		
Units	n/a		
Range	b = 0 (active low), 1 (active high), or X (don't change)		
Default	1		
Response	DRFLVL *DRFLVL1111_1111		
	1DRFLVL *1DRFLVL1		
See Also	[AS], [ASX], DRIVE, DRES, DRFEN, [ER], TAS, TASX, TER		

The Drive Fault Level (DRFLVL) command is used to individually set the fault input level for each axis. To enable the drive fault inputs for each axis, use the DRFEN command (default power-up state is disabled). Use the following table for setting the drive fault level for Compumotor drives.

Compumotor Product	Drive Fault Level
GEMINI, APEX, Dynaserv, LN, OEM Series, S, TQ, ZETA	Active High (DRFLVL1)
SV, BLH, L, LE, PDS, PK130	Active Low (DRFLVLØ)

The drive fault input schematic is shown in your product's *Installation Guide*.

Drive Fault Input Status:

> Use bit #14 in the TAS, TASF, or AS commands to check the status of the drive fault input (if the drive is enabled and the drive fault input is enabled). Bit #4 of the TASX, TASXF, and ASX commands reports the status *even if the drive and the drive fault input are disabled*.

Drive Fault Level (DRFLVL)	Status of device driving the Fault input	AS **bit #14 and** ASX **bit #4**
DRFLVL1 (active high)	OFF or not connected (not sinking current)	1 (drive fault has occurred)
	ON (sinking current)	Ø
DRFLVLØ (active low)	OFF or not connected (not sinking current)	Ø
	ON (sinking current)	1 (drive fault has occurred)

When a drive fault occurs, motion will be stopped on all axes (stopped at the LHAD & LHADA deceleration values) and program execution will be terminated.

Example:
```
DRFLVL0101        ; Set drive fault level to be active low on axes 1 & 3,
                  ; active high on axes 2 & 4
```


DRIVE   Drive Enable

		Product	**Rev**
Type	Drive Configuration	6K	5.0
Syntax	<!><@><a>DRIVE<b><b><b><b><b><b><b><b>		
Units	n/a		
Range	b = 0 (shutdown), 1 (enable), or X (don't change)		
Default	0 (shutdown)		
Response	DRIVE *DRIVE1111_1111		
	1DRIVE *1DRIVE1		
See Also	[AS], [ASS], AXSDEF, DRFEN, DRFLVL, DRES, KDRIVE, TAS, TASX, TER		

The Drive Enable command energizes (DRIVE1) or de-energizes (DRIVEØ) a Compumotor motor/drive combination. The internal shutdown output circuit is illustrated in the product's *Installation Guide*.
NOTE: If the Disable Drive on Kill (KDRIVE) mode is enabled, the drive will be disabled in the event of a kill command or kill input.

Steppers: DRIVE1 energizes the motor drive (Shutdown+ sinks current and Shutdown- sources current).
                  DRIVEØ de-energizes the motor drive (Shutdown+ sources current and Shutdown- sinks current).

Servos: DRIVE1 energizes the motor drive (the SHTNO relay output is connected to COM, and the SHTNC relay output is disconnected from COM). DRIVEØ de-energizes the motor drive (the SHTNO relay output is disconnected from COM, and the SHTNC relay output is connected to COM). DRIVE1 also sets the commanded position (TPC) equal to the actual position (TPE).

NOTE: The DRIVEØ command will not de-energize a motor drive during motion.

Example:
```
DRIVE1110        ; Energize drives 1 through 3, de-energize drive 4
```

DRPCHK   RP240 Check

Type	Communication Interface; Display (RP240) Interface	**Product**	**Rev**
Syntax	<!>DRPCHK<i>	6K	5.0
Units	n/a		
Range	0-3		
Default	0 for port COM1, 3 for port COM2 (PORT command setting determines which COM port's DRPCHK setting is checked)		
Response	DRPCHK *DRPCHK3		
See Also	LOCK, PORT, XONXOFF		

The Remote COM Port Check (DRPCHK) command is used to indicate whether a port is to be used with an RP240 or with 6K language commands. The DRPCHK command affects the COM port selected with the last PORT command. The DRPCHK command value is automatically saved in battery-backed RAM.

NOTE: COM1 is the connector labeled "RS-232" and COM2 is the connector labeled "RS-232/485."

DRPCHKØ......... The serial port will be used for 6K language commands. This is the default setting for COM1, and if using RS-485 half duplex on COM2. Power-up messages appear on all ports set to DRPCHKØ. If you ordered the FieldBus version of the 6K product, COM2 is factory-set to DRPCHKØ.

DRPCHK1......... A check for the presence of an RP240 will be performed at power-up/reset. If an RP240 is present, the 6K product will initialize the RP240. If an RP240 is not present, the port may be used for 6K language commands. Note that RP240 commands will be sent at power-up and reset.

DRPCHK2......... A status check for the presence of an RP240 will be periodically performed (every 5-6 seconds). If an RP240 is plugged in, the 6K product will initialize the RP240. Press F6 on the RP240 periodically until the 6K product recognizes the RP240. (The RP240 indicates that it has been recognized by beeping when F6 is pressed.)

DRPCHK3......... A status check for the presence of an RP240 will be performed at power-up/reset. If an RP240 is present, the 6K product will initialize the RP240. If an RP240 is not present, no commands except DWRITE will have any effect for that port and the COM port will ignore received characters. This is the default setting for COM2, unless you are using RS-485 multi-drop communication (in which case the default changes to DRPCHKØ).

Each port has its own DRPCHK value, but only one may be set to DRPCHK2 or DRPCHK3 at any time.

RS-485 Communication: If you are using RS-485 communication in a multi-drop (requires you to change an internal DIP switch to select half duplex), the default setting for COM2 is DRPCHKØ. If the internal DIP switch setting is left at full duplex, the default setting for COM2 is DRPCHK3.

FieldBus Option: If you ordered the FieldBus version of the 6K product, COM2 is factory-set to DRPCHKØ.

Default values are used until DRPCHK is set for the first time. DRPCHK values are automatically saved in non-volatile memory. They do not change until you set new values. It may be advisable to include the DRPCHK command in your start-up program to ensure that it powers up in the correct setting for your current application.

DSTP   Enable/Disable RP240 Stop Key

Type	Display (RP240) Interface	**Product**	**Rev**
Syntax	`<!>DSTP<b>`	6K	5.0
Units	b = enable bit		
Range	b = 1 (enable) or 0 (disable)		
Default	1 (enable)		
Response	DSTP *DSTP1		
See Also	DLED, [DREAD], [DREADF]		

Use the DSTP command to enable or disable the stop key on the RP240 panel.

DVAR   Display Variable on RP240

Type	Display (RP240) Interface	**Product**	**Rev**
Syntax	`<!>DVARi,<i>,<i>,<i>`	6K	5.0
Units	See below		
Range	n/a		
Default	See below		
Response	n/a		
See Also	[DREAD], [DREADF], DVARB, DVARI, DWRITE, VAR		

The Display Variable on RP240 (DVAR) command is used to display a numeric variable on the RP240's LCD at the current cursor location:

1st `i` = Variable number [Range 1-225]
2nd `i` = Number of whole digits displayed (left of decimal point) [Range 0-9]
3rd `i` = Number of fractional digits displayed (right of decimal point) [Range 0-8]
4th `i` = Sign bit: 0 = no sign displayed, 1 = display + or -

Example：
```
VAR2=542.14      ; Assign the value 542.14 to variable #2
DVAR2,6,3,1      ; Display variable #2 as +000542.140
DVAR2,3,1,0      ; Display variable #2 as 542.1
DVAR2,3,,1       ; Display variable #2 as +542
```

DVARB   Display Binary Variable on RP240

Type	Display (RP240) Interface	**Product**	**Rev**
Syntax	<!>DVARBi,<i>	6K	5.0
Units	See below		
Range	n/a		
Default	See below		
Response	n/a		
See Also	DVAR, DVARI, DWRITE, VARB		

The Display Binary Variable on RP240 (DVARB) command is used to display a binary variable on the
RP240's LCD at the current cursor location:

1st i   =   Variable number [Range 1-125]
2nd i   =   Number of bits displayed [Range 1-32]

Example
```
VARB2=b11001X11 ; Assign the value 11001X11 to binary variable #2
DVARB2,6        ; Display binary variable #2 as 1100_1X
DVARB2,3        ; Display binary variable #2 as 110
DVARB2,1        ; Display binary variable #2 as 1
```

DVARI   Display Integer Variable on RP240

Type	Display (RP240) Interface	**Product**	**Rev**
Syntax	<!>DVARIi,<i>,<i>	6K	5.0
Units	See below		
Range	n/a		
Default	See below		
Response	n/a		
See Also	DVAR, DVARB, DWRITE, VARI		

The Display Integer Variable on RP240 (DVARI) command is used to display an integer variable on the
RP240's LCD at the current cursor location:

1st i   =   Variable number [Range 1-225]
2nd i   =   Number of whole digits displayed [Range 0-9]
3rd i   =   Sign bit: 0=no sign displayed, 1=display + or - sign
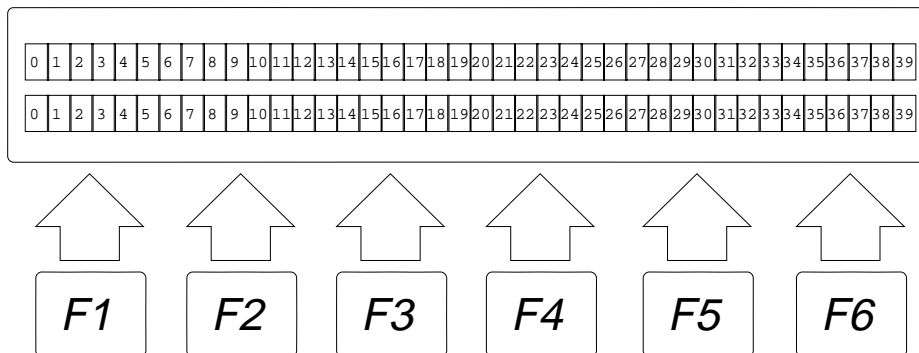
Example
```
VARI2=542       ; Assign the value 542 to integer variable #2
DVARI2,6,1      ; Display integer variable #2 as +000542
DVARI2,3,1      ; Display integer variable #2 as =+542
```

DWRITE   Write Text on RP240

		Product	**Rev**
Type	Display (RP240) Interface	6K	5.0
Syntax	`<!>DWRITE"message"`		
Units	n/a		
Range	Message can be ≤ 80 characters (may not use characters ", \, * or :)		
Default	See below		
Response	n/a		
See Also	DCLEAR, DLED, DPASS, DPCUR, DVAR, DVARB, DVARI, PORT		

The Write Text on RP240 (`DWRITE`) command displays a message on the RP240's LCD starting at the current cursor location. A message is a character string of up to 80 characters in length. The characters within the string may be any characters except quote ("), backslash (\), asterisk (*), and colon (:). Strings that have lower-case letters will be converted to upper case prior to display (see example).

The following graphic shows the location of the RP240's two-line, 40-character display. It also shows the characters in relation to the function keys.

HINT: If you do not have an RP240 and wish to send (only) characters out the serial port to another serial device, you may use this command. Place a backslash (\) before non-alphanumeric characters.
Example: `DWRITE"HOMING SUCCESSFUL\13"` = send message plus <CR>

Example:
```
DCLEAR0                     ; Clear RP240 display
DPCUR1,12                   ; Move cursor to line 1, column 12
DWRITE"Enter Number of Parts"; RP240 will display: ENTER NUMBER OF PARTS
VAR1=DREAD                   ; RP240 waiting for data entry
```

E
Enable Communication

		Product	Rev
Type	Communication Interface	6K	5.0
Syntax	`<i_><!>E<b>`		
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

		Product	Rev
Type	Communication Interface	6K	5.0
Syntax	`<!>ECHO<b>`		
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<b><b><b><b><b><b><b><b>	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**

		Product	**Rev**
Type	`Encoder; Controller Configuration`	6K	5.0
Syntax	`<!><@><a>ENCCNT<b><b><b><b><b><b><b><b>`		
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**

		Product	**Rev**
Type	`Encoder; Controller Configuration`	6K	5.0
Syntax	`<!><@><a>ENCPOL<b><b><b><b><b><b><b><b>`		
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)
```

ENCSND   Encoder Step and Direction Mode

Type	Encoder; Counter	**Product**	**Rev**
Syntax	<!><@><a>ENCSND<b><b><b><b><b><b><b><b>	6K	5.0
Units	n/a		
Range	b = 0 (quadrature signal), 1 (step & direction) or X (don't care)		
Default	0		
Response	ENCSND: *ENCSND0000_0000		
	1ENCSND: *1ENCSND0		
See Also	MESND, [PE], SFB, TPE		

Use the ENCSND 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."

> ENCSND0......(default setting) accept a quadrature signal from an encoder.
>
> ENCSND1......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, Ø, 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 Ø 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=b11Ø1)`, 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 a 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; Ø = 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 `EFAILØ` 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

		Product	Rev
Type	Subroutine or Program Definition	6K	5.0
Syntax	<!>ERASE		
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

		Product	Rev
Type	Encoder Configuration	6K	5.0
Syntax	<!><@><a>ERES<i>,<i>,<i>,<i>,<i>,<i>,<i>,<i>		
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:

<u>Stepper axes</u>:
- -RE, -RC, -EC, and -E Series Encoders: ... `ERES4000`
- -HJ Series Encoders: .... `ERES2048`

<u>Servo axes</u>:
- SM, N or J Series Servo Motors:.............. SM/N/JxxxxD-xxxx: `ERES2000`
  SM/N/JxxxxE-xxxx: `ERES4000`

<u>Dynaserv (stepper or servo)</u>:
- 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, ERRDEF, ERRLVL, ERROK, PORT, TCMDER`		

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.
ERRLVLØ	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.

ERROK   Good Prompt

Type	Communication Interface	**Product**	**Rev**
Syntax	`<!><@>ERROK<i>,<i>,<i>,<i>`	6K	5.0
Units	n/a		
Range	i = 0 – 256		
Default	13,10,62,32		
Response	ERROK: *ERROK13,10,62,32		
See Also	ERRBAD, ERRDEF, ERRLVL, PORT, XONOFF		

The Good Prompt (`ERROK`) command designates the characters to be placed into the output buffer after a command has been entered correctly. 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 greater than symbol is ASCII 62, 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 `ERROK` command.

Example:
```
ERROK13,0,0,0   ; Place a carriage return only in the output buffer after
                ; processing a valid command
```

ERROR   Error-Checking Enable

Type	Error Handling	**Product**	**Rev**
Syntax	`<!><%>ERROR<b><b>...<b><b> (32 bits)`	6K	5.0
Units	n/a		
Range	b = 0 (disable), 1 (enable), or X (don't change)		
Default	0		
Response	ERROR: *ERROR0000_0000_0000_0000_0000_0000_0000_0000		
	bit 1 ⬆ ⬆ bit 32		
See Also	[ASX], DRFEN, DRFLVL, EFAIL, [ER], ERRORP, ESTALL, GOWHEN, INFNC, K, LH, LIMFNC, LS, S, TASX, TER, TRGFN		

When an error-checking bit is enabled (`ERROR11...11`), the operating system will respond to a specific execution error by doing a GOSUB or a GOTO to the error program defined with the `ERRORP` command (see table below). Each bit corresponds to a different error condition. To enable or disable a specific bit, the syntax is `ERROR.n-b`, where "n" is the error bit number and "b" is either 1 to enable or Ø to disable.

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.

Bit #	Function (Error bits #13, #15, and #18 - #32 are reserved.)	Branch Type
1*	Stall Detected: Functions when stall detection has been enabled (`ESTALL`). `ESK` **must be enabled**.	GOSUB
2	Hard Limit Hit: Functions when hard limits are enabled (`LH`).	GOTO if `COMEXLØ`; GOSUB if `COMEXL1`
3	Soft Limit Hit: Functions when soft limits are enabled (`LS`).	GOTO if `COMEXLØ`; GOSUB if `COMEXL1`
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`).	GOTO
5	Commanded Kill or Commanded Stop (a K, !K, `<ctrl>K`, S, or !S command is sent). **NOTE** If you want the program to stop, you must issue the `!HALT` command.	!K = GOTO; !S = GOTO if `COMEXSØ`; !S = GOSUB if `COMEXS1`, but need !C

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 `TCMDER` command).	GOSUB
17	Encoder failure (`EFAIL1` must be enabled before error can be detected; error is cleared by sending `EFAILØ` 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=b1ØX)`). 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.

> 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., <u>2%ERROR.2-1</u> and <u>2%ERRORP FIX</u> for Task <u>2</u>). 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 <u>not</u> 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 (`COMEXC1`). 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 ERRORP	How to Remedy the Error
1	Stepper axes only: Stall detected (Stall Detection and Kill On Stall must be enabled first—see `ESTALL` and `ESK`, respectively)	Gosub	Issue a `GO` command.
2	Hard Limit Hit (hard limits must be enabled first—see `LH`)	If `COMEXLØ`, then Goto; If `COMEXL1`, then Gosub	Change direction & issue `GO` command on the axis that hit the limit; or issue `LHØ`.
3	Soft Limit Hit (soft limits must be enabled first—see `LS`)	If `COMEXLØ`, then Goto; If `COMEXL1`, then Gosub	Change direction & issue `GO` command on the axis that hit the limit; or issue `LSØ`.
4	Drive Fault (Detected only if drive enabled – `DRIVE`, drive fault input enabled – `DRFEN`, and drive fault level correct – `DRFLVL`).	Goto	Clear the fault condition at the drive, & issue a `DRIVE1` command for the faulted axis.
5	Commanded Stop or Kill (whenever a K, !K, <ctrl>K, S, or !S command is sent) — See "Commanded Kill or Stop" note below.	If `!K`, then Goto; If `!S` & `COMEXSØ`, then Goto; If `!S` & `COMEXS1`, then Gosub, but need `!C`	No fault condition is present—there is no error to clear. ┌─────────────────────────────┐ │ If you want the program to stop, you must │ │ issue the `!HALT` command. │ └─────────────────────────────┘
6	Kill Input Activated (see `INFNCi-C` or `LIMFNCi-C`)	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: `STRGTEØ, DØ, 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 `TCMDER` command.
17	Encoder failure (`EFAIL1` must be enabled before error can be detected).	Gosub	Send the `EFAILØ` 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 <u>not</u> 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

			Product	Rev
Type	`Encoder Configuration`		**6K**	5.0
Syntax	`<!><@><a>ESDB<i>,<i>,<i>,<i>,<i>,<i>,<i>,<i>`			
Units	`i = encoder steps`			
Range	`0 - 99,999,999`		(applicable only to	
Default	`0`		stepper axes)	
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

			Product	Rev
Type	`Encoder Configuration`		**6K**	5.0
Syntax	`<!><@><a>ESK<b><b><b><b><b><b><b><b>`			
Units	`n/a`			
Range	`b = 0 (disable), 1 (enable), or X (don't change)`		(applicable only to	
Default	`0`		stepper axes)	
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

			Product	Rev
Type	`Encoder Configuration`		**6K**	5.0
Syntax	`<!><@><a>ESTALL<b><b><b><b><b><b><b><b>`			
Units	`n/a`			
Range	`b = 0 (disable), 1 (enable), or X (don't change)`		(applicable only to	
Default	`0`		stepper axes)	
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 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) – *default setting*
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>,<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 <u>not</u> 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

			Product	Rev
Type	Following		6K	5.0
Syntax	`<!><@><a>FMAXA<r>,<r>,<r>,<r>,<r>,<r>,<r>,<r>`			
Units	r = units/sec/sec			
Range	r = 0.00001 - 39,999,998 (scalable with SCLA)		(applicable only to	
Default	0.00 (no limit imposed)		stepper axes)	
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

			Product	Rev
Type	Following		6K	5.0
Syntax	`<!><@><a>FMAXV<r>,<r>,<r>,<r>,<r>,<r>,<r>,<r>`			
Units	r = units/sec (scalable with SCLV)			
Range	r = 0.000000-1600000.000000		(applicable only to	
Default	0.00 (no limit imposed)		stepper axes)	
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.
```

FMCLEN   Master Cycle Length

Type	Following	**Product**	**Rev**
Syntax	`<!><@><a>FMCLEN<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	`FMCLEN    *FMCLEN0,0,0,0,0,0,0,0`		
	`1FMCLEN  *1FMCLEN0`		
See Also	`FMCNEW, FMCP, FOLEN, [ FS ], GOWHEN, [ PMAS ], SCLMAS, TFS,`		
	`TPMAS, WAIT`		

The `FMCLEN` 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 `FMCLEN` is zero (`FMCLENØ`), 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, `FMCLEN`, 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
FMCLEN3,(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<b><b><b><b><b><b><b><b>`	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	`FMCLEN, 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 (`FMCLEN`) 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

		Product	Rev
Type	Following	**6K**	**5.0**
Syntax	`<!><@><a>FMCP<r>,<r>,<r>,<r>,<r>,<r>,<r>,<r>`		
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,+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 (`FMCPØ`), 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

		Product	Rev
Type	Following	**6K**	**5.0**
Syntax	`<!><@><a>FOLEN<b><b><b><b><b><b><b><b>`		
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, `MC1`, 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
FMCLEN100       ; 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
FMCLEN40          ; 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
MC0               ; 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<b><b><b><b><b><b><b><b>`		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 (<u>follower's commanded position loses synchronization with the master</u>).

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<±ii>,<±ii>,<±ii>,<±ii>,<±ii>,<±ii>,<±ii>,<±ii>`		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 (`±ii`) configures that axis as a follower following the specified master count source. In the syntax for each follower axis (`±ii`), 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 place int he data field (`±ii`), that axis will follow the counts from the Master Encoder (the separate encoder labeled "MASTER ENCODER").
- If a zero (∅) is placed in the data field (`±ii`), 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 user 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

		Product	Rev
Type	Following	6K	5.0
Syntax	<!><@><a>FOLMD<r>,<r>,<r>,<r>,<r>,<r>,<r>,<r>		
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
```

www.comoso.com

FOLRD

Denominator of Follower-to-Master Ratio

		Product	**Rev**
Type	Following	6K	5.0
Syntax	`<!><@><a>FOLRD<r>,<r>,<r>,<r>,<r>,<r>,<r>,<r>`		
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

		Product	**Rev**
Type	Following	6K	5.0
Syntax	`<!><@><a>FOLRN<r>,<r>,<r>,<r>,<r>,<r>,<r>,<r>`		
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		
Syntax	<!><@><a>FOLRNF<r>,<r>,<r>,<r>,<r>,<r>,<r>,<r>	6K	5.0
Units	r = follower distance in steps		
Range	0.00000		
Default	0		
Response	FOLRNF *FOLRNF0,0,0,0,0,0,0,0		
	1FOLRNF *1FOLRNF0		
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 Ø, 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		
Syntax	<!><@><a>FPPEN<b><b><b><b><b><b><b><b>	6K	5.0
Units	n/a		
Range	b = 0 (disable), 1 (enable) or X (don't change)		
Default	1		
Response	FPPEN *FPPEN1111_1111		
	1FPPEN *1FPPEN1		
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 `FPPENØ` 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 (`FPPENØ`) 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:
```
FPPEN11          ; 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, FMCLEN, 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 = ∅)	
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–`MC∅`).
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 `FMCLEN` 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 (`FMCLEN`) 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=b11Ø1)`, 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, Ø, 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 Ø 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>`	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 (Ø) 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
FSHFC0            ; 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 (∅) 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-2ØØ        ; 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,+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<b><b><b><b><b><b><b><b>`	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`).

<div style="border:1px solid black; padding:10px;">

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*.

</div>

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<b><b><b><b><b><b><b><b>`	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
```


The resulting profile from this program:
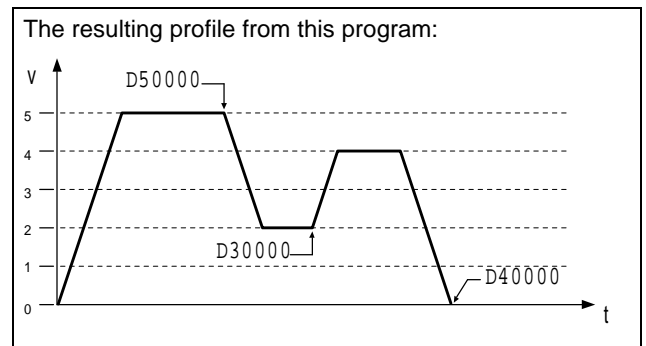
GOL **Initiate Linear Interpolated Motion**

Type	Motion (Linear Interpolated)	**Product**	**Rev**
Syntax	`<!><@>GOL<b><b><b><b><b><b><b><b>`	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/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

		Product	**Rev**
Type	Program; Subroutine Definition; Program Flow Control	6K	5.0
Syntax	`<!>GOTO<t>`		
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		

Use 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 (`MC1`) 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:**


```
GOWHEN (expression),(expression),(expression),  . . .
         Axis 1        Axis 2        Axis 3
```

Relational Expression Syntax:
(`<left operand>` `<relational operator>` `<right operand>`)

Possible Operators:

FB .... Feedback device position
LIM ..... Limit input state
NMCY ... Master cycle number
PC .... Commanded position
PE .... Encoder position
PMAS ... Master position
PSLC ... Slave position
PSHF ... Following shift
IN .... Input state
T .... Dwell (in milliseconds)

Possible Operators:

>=
<=
=
>
<

Possible Operators:

• Numeric variables
  (VAR or VARI)
• Decimal constant
• Binary value (b___)
  for IN operator only

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). <u>See bit #26.</u>

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 <u>only</u> if you enable error-checking bit #14 with the ERROR command (e.g., ERROR.14-1). NOTE: When the error occurs, the controller with 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). <u>See bit #14.</u>

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 progam
```

HALT   Terminate Program Execution

Type	Program Flow Control	**Product**	**Rev**
Syntax	`<!>HALT`	6K	5.0
Units	n/a		
Range	n/a		
Default	n/a		
Response	n/a		
See Also	BP, BREAK, C, ELSE, IF, K, NIF, NWHILE, PS, REPEAT, S, T, UNTIL, WAIT, WHILE		

The Terminate Program Execution (`HALT`) command terminates program execution when processed. This command allows the user to terminate command processing at any point in a program. The programmer may want processing to stop because of an error condition, an input, a variable, or just after a specific motion has been accomplished. This command is useful when debugging a program.

Example:
```
DEF prog1        ; Define a program called prog1
GO1000           ; Initiate motion on axis 1
GOSUB prog2      ; Gosub to subroutine named prog2
GO0100           ; Initiate motion on axis 2
END              ; End program definition
DEF prog2        ; Define a program called prog2
GO1110           ; Initiate motion on axes 1, 2, and 3
IF(IN=b1X0)      ; If onboard input 1 is active (1), and input 3 is inactive (0)
HALT             ; If condition is true break out of program
ELSE             ; Else part of if condition
TPE              ; If condition does not come true transfer position of all
                 ; encoders to PC
NIF              ; End If statement
END              ; End program definition
RUN prog1        ; Execute program prog2
;
; Upon completion of motion on axis 1, subroutine prog2 is called.
; If inputs 1 and 3 are in the correct state after the motion is complete,
; program processing will be terminated. In other words, all commands waiting
; to be parsed in the program buffer will be eliminated.
; **** Note: There will not be a return to prog1.
```

HELP   Technical Support

Type	Program Debug Tool	**Product**	**Rev**
Syntax	`<!>HELP`	6K	5.0
Units	n/a		
Range	n/a		
Default	n/a		
Response	See description below		
See Also	None		

The (`HELP`) command provides the telephone numbers for technical support.

HOM   Go Home

			Product	**Rev**
Type	`Homing`			
Syntax	`<!><@>HOM<b><b><b><b><b><b><b><b>`		6K	5.0
Units	`n/a`			
Range	`b = 0 (home in positive direction), 1 (home in negative direction),` `or X (do not home)`			
Default	`X`			
Response	`n/a`			
See Also	`[ AS ], HOMA, HOMAA, HOMAD, HOMADA, HOMBAC, HOMDF, HOMEDG,` `HOMV, HOMVF, HOMZ, [ LIM ], LIMEN, LIMLVL, PSET, TAS, TLIM`			

The Go Home (`HOM`) command instructs the controller to search for the home position in the direction, and on the axes, specified by the command. If an end-of-travel limit is activated while searching for the home limit, the controller will reverse direction and search for home in the opposite direction. However, if a second end-of-travel limit is encountered, after the change of direction, the homing operation will be aborted.

The status of the homing operation is provided by bit 5 of each axis status register (refer to the `TAS` or `AS` command). *When the homing operation is successfully completed, the absolute position register is set to zero (equivalent to `PSET0`).*

NOTE

Pause and resume functions are not recommended during the homing operation. A Pause command or input will pause the homing motion; however, when the subsequent Resume command or input occurs, motion will resume at the beginning of the homing motion sequence.

The homing operation has several parameters that determine the homing algorithm:

- Home acceleration (`HOMA` and `HOMAA`)
- Home deceleration (`HOMAD` and `HOMADA`)
- Home velocity (`HOMV`)
- Final home velocity (`HOMVF`)
- Home reference edge (`HOMEDG`)
- Backup to home (`HOMBAC`)
- Final home direction (`HOMDF`)
- Active state of home input (`LIMLVL`)
- Home to encoder Z-channel (`HOMZ`)

For more information on homing refer to the *Homing* section of the *Programmer's Guide*.

Example:
```
SCALE1             ; Enable scaling
SCLA25000,25000,1,1 ; Set accel. scaling: axes 1 & 2 = 25000 steps/unit/unit;
                   ; axes 3 & 4 = 1 step/unit/unit
SCLV25000,25000,1,1 ; Set vel. scaling: axes 1 & 2 = 25000 steps/unit;
                   ; axes 3 & 4 = 1 step/unit
@SCLD1             ; Set distance scaling factor for all axes to 1 step/unit
DEL Homrdy         ; Delete program called Homrdy
DEF Homrdy         ; Begin definition of program called Homrdy
@MA0               ; Incremental index mode for all axes
@MC0               ; Preset index mode for all axes
HOMA10,12,1,2      ; Set home acceleration to 10, 12, 1, & 2 units/sec/sec for
                   ; axes 1, 2, 3 & 4
@HOMAD20           ; Set home deceleration to 20 units/sec/sec for all axes
HOMBAC1100         ; Enable backup to home switch on axes 1 and 2 only
HOMEDG0011         ; Axes 1 & 2 stop on the positive-direction edge of the home
                   ; switch, axes 3 and 4 are to stop on negative-direction side
@HOMDF0            ; Set final home direction to positive on all axes.
@HOMZ0             ; Disable homing to encoder Z-channel on all axes
LIMLVLxx0xx0xx0xx0 ; Set home active level to low on axes 1-4
HOMV1,1,1,2        ; Set home velocity to 1, 1, 1, and 2 units/sec for
                   ; axes 1, 2, 3 & 4
@HOMVF.1           ; Sets home final velocity to 0.1 units/sec for all axes
HOM01XX            ; Execute go home in positive-direction on axis 1,
                   ; negative-direction on axis 2. Do not home on axes 3 and 4.
END                ; End definition of Homrdy
```

HOMA — Home Acceleration

Type	Homing	**Product**	**Rev**
Syntax	`<!><@><a>HOMA<r>,<r>,<r>,<r>,<r>,<r>,<r>,<r>`	6K	5.0
Units	`r = units/sec/sec`		
Range	0.00001 – 39,999,998 (depending on the scaling factor)		
Default	10.0000		
Response	`HOMA:    *HOMA10.0000,10.0000,10.0000,10.0000 ...`		
	`1HOMA:   *1HOMA10.0000`		
See Also	`HOM, HOMAD, HOMBAC, HOMDF, HOMEDG, HOMV, HOMVF, HOMZ, [ LIM ],` `LIMEN, LIMLVL, SCALE, SCLA`		

The Home Acceleration (`HOMA`) command specifies the acceleration rate to be used upon executing the next go home (`HOM`) command.

> **UNITS OF MEASURE** and **SCALING**: refer to page 16.

The homing acceleration remains set until you change it with a subsequent homing acceleration command. Homing accelerations outside the valid range are flagged as an error, with a message `*INVALID DATA-FIELD x`, where x is the field number. When an invalid homing acceleration is entered the previous homing acceleration value is retained.

If the home deceleration (`HOMAD`) command has not been entered, the home acceleration (`HOMA`) command will set the home deceleration rate. Once the home deceleration (`HOMAD`) command has been entered, the home acceleration (`HOMA`) command no longer affects home deceleration.

Example: Refer to the go home (`HOM`) command example.

HOMAA — Homing Average Acceleration

Type	Motion (S-Curve)	**Product**	**Rev**
Syntax	`<!><@><a>HOMAA<r>,<r>,<r>,<r>,<r>,<r>,<r>,<r>`	6K	5.0
Units	`r = units/sec/sec`		
Range	0.00001 – 39,999,998 (depending on the scaling factor)		
Default	10.00 (trapezoidal profiling is default, where HOMAA tracks HOMA)		
Response	`HOMAA:    *HOMAA10.0000,10.0000,10.0000,10.0000 ...`		
	`1HOMAA:   *1HOMAA10.0000`		
See Also	`A, AD, ADA, HOM, HOMA, HOMAD, HOMADA, HOMBAC, SCALE, SCLA`		

The Homing Average Acceleration (`HOMAA`) command allows you to specify the average acceleration for an S-curve homing profile. S-curve profiling provides smoother motion control by reducing the rate of change in acceleration and deceleration; this accel/decel rate of change is known as *jerk*. Refer to page 13 for details on S-curve profiling.

Scaling (`SCLA`) affects `HOMAA` the same as it does for `HOMA`. Refer to page 16 for details on scaling.

Example:
```
SCALE0              ; Disable scaling
DEL proge           ; Delete program called proge
DEF proge           ; Begin definition of program called proge
@MA0                ; Select incremental positioning mode
HOMA10,10           ; Set homing max. accel to 10 rev/sec/sec (axes 1 and 2)
HOMAA5,10           ; Set homing avg. accel to 5 rev/sec/sec on axis 1,
                    ; and 10 rev/sec/sec on axis 2
HOMAD10,10          ; Set homing max. decel to 10 rev/sec/sec (axes 1 and 2)
HOMADA5,10          ; Set homing avg. decel to 5 rev/sec/sec on axis 1,
                    ; and 10 rev/sec/sec on axis 2
HOM11XX             ; Execute negative-direction homing moves on axes 1 and 2
; Axis 1 executes a pure S-curve; axis 2 executes a trapezoidal profile.
END                 ; End definition of program
```

HOMAD   Home Deceleration

		Product	**Rev**
Type	Homing	6K	5.0
Syntax	`<!><@><a>HOMAD<r>,<r>,<r>,<r>,<r>,<r>,<r>,<r>`		
Units	`r = units/sec/sec`		
Range	0.00001 - 39,999,998 (depending on the scaling factor)		
Default	10.0000 (HOMAD tracks HOMA)		
Response	HOMAD: *HOMAD10.0000,10.0000,10.0000,10.0000 ...		
	1HOMAD: *1HOMAD10.0000		
See Also	HOM, HOMA, HOMAA, HOMADA, HOMBAC, HOMEDG, HOMDF, HOMV, HOMVF, HOMZ, [LIM], LIMEN, LIMLVL, SCALE, SCLA		

The Home Deceleration (`HOMAD`) command specifies the deceleration rate to be used upon executing the next go home (`HOM`) command.

UNITS OF MEASURE and **SCALING**: refer to page 16.

The home deceleration remains set until you change it with a subsequent home deceleration command. Decelerations outside the valid range are flagged as an error, with a message `*INVALID DATA-FIELD x`, where x is the field number. When an invalid deceleration is entered the previous deceleration value is retained.

If the home deceleration (`HOMAD`) command has not been entered, the home acceleration (`HOMA`) command will set the deceleration rate. Once the home deceleration (`HOMAD`) command has been entered, the home acceleration (`HOMA`) command no longer affects home deceleration. If the `HOMAD` command is set to zero (`HOMADØ`), then the homing deceleration will once again track whatever the `HOMA` command is set to.

Example: Refer to the go home (`HOM`) command example.

HOMADA   Homing Average Deceleration

		Product	**Rev**
Type	Motion (S-Curve)	6K	5.0
Syntax	`<!><@><a>HOMADA<r>,<r>,<r>,<r>,<r>,<r>,<r>,<r>`		
Units	`r = units/sec/sec`		
Range	0.00001 - 39,999,998 (depending on the scaling factor)		
Default	10.00 (HOMADA tracks HOMAA)		
Response	HOMADA: *HOMADA10.0000,10.0000,10.0000,10.0000 ...		
	1HOMADA: *1HOMADA10.0000		
See Also	A, AD, HOM, HOMA, HOMAA, HOMAD, SCALE, SCLA		

The Homing Average Deceleration (`HOMADA`) command allows you to specify the average deceleration for an S-curve homing profile. S-curve profiling provides smoother motion control by reducing the rate of change in acceleration and deceleration; this accel/decel rate of change is known as *jerk*. Refer to page 13 for details on S-curve profiling.

Scaling (`SCLA`) affects `HOMADA` the same as it does for `HOMAD`. Refer to page 16 for details on scaling.

Example:
```
SCALE0              ; Disable scaling
DEL proge           ; Delete program called proge
DEF proge           ; Begin definition of program called proge
@MA0                ; Select incremental positioning mode
HOMA10,10           ; Set homing max. accel to 10 rev/sec/sec (axes 1 and 2)
HOMAA5,10           ; Set homing avg. accel to 5 rev/sec/sec on axis 1,
                    ; and 10 rev/sec/sec on axis 2
HOMAD10,10          ; Set homing max. decel to 10 rev/sec/sec (axes 1 and 2)
HOMADA5,10          ; Set homing avg. decel to 5 rev/sec/sec on axis 1,
                    ; and 10 rev/sec/sec on axis 2
HOM11XX             ; Execute negative-direction homing moves on axes 1 and 2.
                    ; Axis 1 executes a pure S-curve.
                    ; Axis 2 executes a trapezoidal profile.
END                 ; End definition of program
```

HOMBAC   Home Backup Enable

Type	Homing	**Product**	**Rev**
Syntax	`<!><@><a>HOMBAC<b><b><b><b><b><b><b><b>`	6K	5.0
Units	n/a		
Range	b = 0 (disable), 1 (enable), or X (don't change)		
Default	0		
Response	HOMBAC: *HOMBAC0000_0000		
	1HOMBAC: *1HOMBAC0		
See Also	HOM, HOMA, HOMAA, HOMAD, HOMADA, HOMDF, HOMEDG, HOMV, HOMVF, HOMZ, [LIM], LIMEN, LIMLVL		

The Home Backup Enable (HOMBAC) command enables or disables the backup to home switch function. When this function is enabled, the motor will decelerate to a stop after encountering the active edge of the home region, and then move the motor in the opposite direction at the home final velocity (HOMVF) until the active edge of the home region is encountered. This motion will occur regardless of whether or not the home input is active at the end of the deceleration of the initial go home move.

Example: Refer to the go home (HOM) command example.

HOMDF   Home Final Direction

Type	Homing	**Product**	**Rev**
Syntax	`<!><@><a>HOMDF<b><b><b><b><b><b><b><b>`	6K	5.0
Units	n/a		
Range	b = 0 (positive-direction), 1 (negative-direction), or X (don't change)		
Default	0		
Response	HOMDF: *HOMDF0000_0000		
	1HOMDF: *1HOMDF0		
See Also	HOM, HOMA, HOMAA, HOMAD, HOMADA, HOMBAC, HOMEDG, HOMV, HOMVF, HOMZ, [LIM], LIMEN, LIMLVL		

The Home Final Direction (HOMDF) command specifies the direction the 6K Series product is to be traveling when the home algorithm does its final approach. This command is operational when backup to home (HOMBAC) is enabled, or when homing to an encoder Z channel (HOMZ).

Example: Refer to the go home (HOM) command example.

HOMEDG   Home Reference Edge

Type	Homing		**Product**	**Rev**
Syntax	`<!><@><a>HOMEDG<b><b><b><b><b><b><b><b>`		6K	5.0
Units	n/a			
Range	b = 0 (positive-direction edge), 1 (negative-direction edge),			
	or X (don't change)			
Default	0			
Response	HOMEDG: *HOMEDG0000_0000			
	1HOMEDG: *1HOMEDG0			
See Also	HOM, HOMA, HOMAA, HOMAD, HOMADA, HOMBAC, HOMDF, HOMV, HOMVF,			
	HOMZ, [LIM], LIMEN, LIMLVL,			

The Home Reference Edge (HOMEDG) command specifies which edge of the home switch the homing operation will consider as its final destination.

As illustrated below, the positive-direction edge of the home switch is defined as the first switch transition seen by the controller when traveling off of the positive-direction end-of-travel limit in the negative direction. The negative-direction edge of the home switch is defined as the first switch transition seen by the indexer when traveling off of the negative-direction end-of-travel limit in the positive-direction. This command is operational when backup to home (HOMBAC) is enabled.
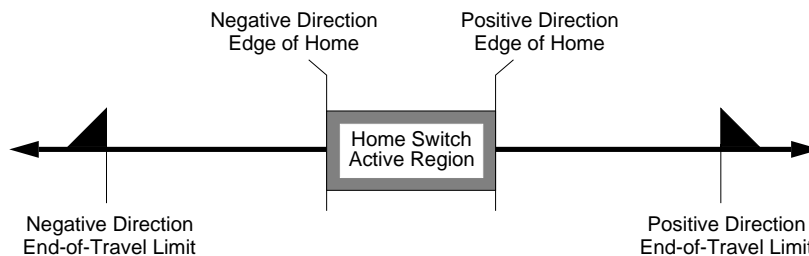
Example: Refer to the go home (HOM) command example.

HOMV   Home Velocity

Type	Homing		**Product**	**Rev**
Syntax	`<!><@><a>HOMV<r>,<r>,<r>,<r>,<r>,<r>,<r>,<r>`		6K	5.0
Units	r = units/sec (scalable with SCLV)			
Range	Stepper Axes: 0.00000-2,048,000 (max. depends on SCLV & PULSE)			
	Servo Axes: 0.00000-6,500,000 (max. depends on SCLV)			
Default	1.0000			
Response	HOMV: *HOMV1.0000,1.0000,1.0000,1.0000 ...			
	1HOMV: *1HOMV1.0000			
See Also	HOM, HOMA, HOMAA, HOMAD, HOMADA, HOMBAC, HOMDF, HOMEDG, HOMVF,			
	HOMZ, [LIM], LIMEN, LIMLVL, PULSE, SCALE, SCLV			

The Home Velocity (HOMV) command specifies the velocity to use when the home algorithm begins its initial go home (HOM) move. The velocity remains set until you change it with a subsequent home velocity command. Velocities outside the valid range are flagged as an error, with a message *INVALID DATA-FIELD x, where x is the field number. When an invalid velocity is entered the previous velocity value is retained.

UNITS OF MEASURE and **SCALING**: refer to page 16.

Example: Refer to the go home (HOM) command example.

HOMVF   Home Final Velocity

Type	Homing	**Product**	**Rev**
Syntax	`<!><@><a>HOMVF<r>,<r>,<r>,<r>,<r>,<r>,<r>,<r>`	6K	5.0
Units	r = units/sec (scalable with SCLV)		
Range	Stepper Axes: 0.00000-2,048,000 (max. depends on SCLV & PULSE)		
	Servo Axes: 0.00000-6,500,000 (max. depends on SCLV)		
Default	0.1000		
Response	HOMVF: *HOMVF0.1000,0.1000,0.1000,0.1000 ...		
	1HOMVF: *1HOMVF0.1000		
See Also	HOM, HOMA, HOMAA, HOMAD, HOMADA, HOMBAC, HOMDF, HOMEDG, HOMV,		
	HOMZ, [LIM], LIMEN, LIMLVL, PULSE, SCALE, SCLV		

The Home Final Velocity (HOMVF) command specifies the velocity to use when the home algorithm does its final approach. This command is only operational when backup to home (HOMBAC) is enabled, or when homing to an encoder Z channel (HOMZ).

The velocity remains set until you change it with a subsequent home final velocity command. Velocities outside the valid range are flagged as an error, with a message *INVALID DATA-FIELD x, where x is the field number. When an invalid velocity is entered, the previous velocity value is retained.

UNITS OF MEASURE and **SCALING**: refer to page 16.

Example: Refer to the go home (HOM) command example.

HOMZ   Home to Encoder Z-channel Enable

Type	Homing	**Product**	**Rev**
Syntax	`<!><@><a>HOMZ<b><b><b><b><b><b><b><b>`	6K	5.0
Units	n/a		
Range	b = 0 (disable), 1 (enable), or X (don't change)		
Default	0		
Response	HOMZ: *HOMZ0000_0000		
	1HOMZ: *1HOMZ0		
See Also	[ASX], HOM, HOMA, HOMAA, HOMAD, HOMADA, HOMBAC, HOMDF,		
	HOMEDG, HOMV, HOMVF, [LIM], LIMEN, LIMLVL, TASX		

This command enables homing to an encoder z-channel after the initial home input has gone active. NOTE: The home limit input is required to go active prior to homing to the Z channel. The state of the Z-channel is reported with bit 6 of the ASX and TASX register.

Example: Refer to the go home (HOM) command example.

IF( )   IF Statement

Type	Program Flow Control or Conditional Branching	**Product**	**Rev**
Syntax	`<!>IF(expression)`	6K	5.0
Units	n/a		
Range	Up to 80 characters (including parentheses)		
Default	n/a		
Response	n/a		
See Also	ELSE, NIF		

This command is used in conjunction with the `ELSE` and `NIF` commands to provide conditional branching. If the expression contained within the parenthesis of the `IF` command evaluates true, then the commands between the `IF` and the `NIF` are executed. If the expression evaluates false, the commands between the `IF` and the `NIF` are ignored, and command processing continues with the first command following the `NIF`.

When the `ELSE` command is used in conjunction with the `IF` command, true `IF` evaluations cause the commands between the `IF` and `ELSE` commands to be executed, the commands after the `ELSE` until the `NIF` are ignored. False `IF` evaluations cause commands between the `ELSE` and the `NIF` to be executed, with commands between the `IF` and the `ELSE` ignored. The `ELSE` command is optional and does not have to be included in the `IF` statement.

The `IF( ).. ELSE .. NIF` structure can be nested up to 16 levels deep.

NOTE: Be careful about performing a `GOTO` between `IF` and `NIF`. Branching to a different location within the same program will cause the next `IF` statement encountered to be nested within the previous `IF` statement, unless an `NIF` command has already been encountered.

IF statement programming order:
```
IF(expression)...commands...NIF
or
IF(expression)...commands...ELSE...commands...NIF
```

All logical operators (`AND`, `OR`, `NOT`), and all relational operators (=, >, >=, <, <=, <>) can be used within the `IF` expression. There is no limit on the number of logical operators, or on the number of relational operators allowed within a single `IF` expression. The limiting factor for the `IF` expression is the command length. **The total character count for the `IF` command and expression cannot exceed 80 characters.** (e.g., If you add up the letters in the `IF` command and the letters within the ( ) expression, including the parenthesis and excluding each space, this count must be less than or equal to 80.)

All assignment operators (`A`, `AD`, `AS`, `ASX`, `D`, `ER`, `IN`, `LIM`, `MOV`, `OUT`, `PC`, `PCE`, `PCM`, `PE`, `PER`, `PMAS`, `SEG`, `SS`, `TIM`, `US`, `V`, `VEL`, `VELA`, etc.) can be used within the `IF` expression.

Multiple parentheses may not be used within the `IF` command.

Example:
```
IF(IN=b1X0 AND VAR1=1)      ; If onboard input 1 is ON, input 3 is OFF, and
                            ; variable 1 equals 1, then the IF statement evaluates
                            ; true, so commands between this statement and NIF
                            ; are executed
  TREV                      ; Transfer revision level
  NIF                       ; End IF statement
IF(1A<5000 AND 2PC>50000)   ; If the acceleration of axis 1 is less than 5000, and
                            ; the commanded position of axis 2 is greater than
                            ; 50000, then do the IF statement. Note: The
                            ; acceleration value used is programmed acceleration,
                            ; not actual.
  VAR1=VAR1+1               ; Increment variable 1
  NIF                       ; End if statement
IF(4VEL<123 OR 4VEL>156)    ; If the current velocity of axis 4 is less than 123
                            ; or if it is greater than 156, then do the commands
                            ; following the IF statement
  WRITE"Something's Wrong\13" ; Put message Something's Wrong<cr> in output buffer
  NIF                       ; End if statement
IF(OUT=b110X1 AND VAR1<=13) ; If onboard outputs 1, 2 and 5 are ON, output 3 is
                            ; off and variable 1 is less than or equal to 13,
                            ; then set variable 1 equal to variable 1 plus 1,
                            ; else set variable 1 equal to variable 1 minus 1
  VAR1=VAR1+1
  ELSE
  VAR1=VAR1-1
  NIF                       ; End IF statement
```

[IN]   Input Status

Type	Assignment or Comparison
Syntax	See below
Units	n/a
Range	n/a
Default	n/a
Response	n/a
See Also	GOWHEN, INFNC, [LIM], ONIN, TIN, VARB

Product	Rev
6K	5.0

Use the IN operator is used to assign the input value to a binary variable (VARB), 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, Ø, 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 Ø through 9.

Syntax: VARBn=IN where "n" is the binary variable number, or IN can be used in an expression such as IF(IN=b11Ø1), or IF(IN=h7F). To assign only one input value to a binary variable, instead of all the inputs, the bit select (.) operator can be used. For example, VARB1=2IN.10 assigns the binary state of input 10 (2nd pin on SIM 2) on I/O brick 2 to binary variable 1.

The number of inputs available for assignment or comparison varies from one 6K Series product to another; to ascertain the input bit assignments for your 6K Series product refer to page 6. The function of the inputs is established with the INFNC command (although the IN operator looks at any trigger or external digital input, regardless of its assigned function from the INFNC command).

Example:
```
VARB1=3IN          ; Input status on I/O brick 3 assigned to binary variable 1
VARB2=2IN.12       ; Input bit 12 on I/O brick 2 assigned to binary variable 2
VARB2              ; Response if bit 12 is set to 1:
                   ; *VARB2=XXXX_XXXX_XXX1_XXXX_XXXX_XXXX_XXXX_XXXX
IF(1IN=b111011X11); If the input status contains 1's for inputs 1,2,3,5,6,8,& 9,
                   ; and 0 for input 4 on I/O brick 1, do the commands
                   ; following the IF statement
TREV               ; Transfer revision level
NIF                ; End IF statement
IF(2IN=hEF00)      ; If the input status contains 1's for I/O brick 2's inputs
                   ; 1,2,3,5,6,7,& 8, and 0's for every other input, do the
                   ; commands following the IF statement
TREV               ; Transfer revision level
NIF                ; End IF statement
```

INDEB   Input Debounce Time

Type	Input	**Product**	**Rev**
Syntax	`<!><B>INDEB<i>`	6K	5.0
Units	i = time in milliseconds (ms)		
Range	i = 2-250		
Default	4		
Response	INDEB: *0INDEB4		
	1INDEB: *1INDEB4		
See Also	INFNC, LIMFNC, RE, REG, TIN, TLIM, TRGFN, TRGLOT		

The INDEB command governs the debounce time for all of the inputs on the specified I/O brick (all trigger inputs, found on the "TRIGGERS/OUTPUTS" connectors, are collectively considered I/O brick 0). The debounce is the period of time that the input must be held in a certain state before the controller recognizes it. This directly affects the rate at which the inputs can change state and be recognized. The default setting is 4 ms.

Exception for Trigger Inputs: For trigger inputs that are assigned the "Trigger Interrupt" function (INFNCi-H), the debounce is instead governed by the TRGLOT setting. The TRGLOT setting applies to all trigger inputs defined as "Trigger Interrupt" inputs. The TRGLOT debounce time is the time required between a trigger's initial active transition and its secondary active transition. This allows rapid recognition of a trigger, but prevents subsequent bouncing of the input from causing a false position capture. The default setting is 24 ms.

Limit Inputs. The limit inputs found on the "LIMITS/HOME" connectors are not normally debounced; however, if a limit is assigned a different function with the LIMFNC command (other than LIMFNCi-R, LIMFNCi-S, or LIMFNCi-T), the input is debounced using the INDEB setting for the on-board trigger inputs (I/O brick 0). If a general-purpose input or trigger input is assigned a limit input function (INFNCi-R, INFNCi-S, or INFNCi-T), the input will not be debounced.

Example:
```
INDEB6          ; Assign all onboard trigger a debounce time of 6 ms
2INDEB10        ; Assign inputs on I/O brick 2 a debounce time of 10 ms
1INDEB12        ; Assign inputs on I/O brick 1 a debounce time of 12 ms
```

INDUSE   Enable/Disable User Status

Type	Controller Configuration	**Product**	**Rev**
Syntax	`<!>INDUSE<b>`	6K	5.0
Units	n/a		
Range	b = 0 (disable) or 1 (enable)		
Default	0		
Response	INDUSE: *INDUSE0		
See Also	INDUST, ONUS, TUS, [US]		

The Enable/Disable User Status (INDUSE) command enables the INDUST command updates. When this command is not enabled, the user status bits (INDUST) can be defined; however, they will not be updated in the US or the TUS commands until INDUSE is enabled.

Example:
```
INDUSE1         ; Enable user status
```

INDUST   User Status Definition

		Product	**Rev**
Type	Controller Configuration	6K	5.0
Syntax	`<!><B>INDUST<i><-<i><c>>`		
Units	See description below		
Range	1st i = 1 - 16; 2nd i = 1 - 32; c = A through S		
Default	See description below		

Response	INDUST: *INDUST1-1A AXIS 1 STATUS - STATUS OFF
	               (...repeated for all 16 user status bits...)
	               *INDUST16-4D AXIS 4 STATUS - STATUS OFF
	INDUST1: *INDUST1-1A AXIS 1 STATUS - STATUS OFF		

See Also	[AS], [ASX], [IN], INDUSE, ONUS, [SS], TAS, TASX, TIN, TSS, TUS, [US]		

The User Status Definition (INDUST) command establishes the user status bit function. Each bit can correspond to an axis status bit, a system status bit, an input, an interrupt bit, or an extended axis status bit. The default for each user status bit is as follows:

Default for the 6K product (first two AS status bits for each axis):

Bits 1-2 = first 2 bits of axis status (AS) for axis 1
Bits 3-4 = first 2 bits of axis status (AS) for axis 2
Bits 5-6 = first 2 bits of axis status (AS) for axis 3
Bits 7-8 = first 2 bits of axis status (AS) for axis 4
Bits 9-10 = first 2 bits of axis status (AS) for axis 5
Bits 11-12 = first 2 bits of axis status (AS) for axis 6
Bits 13-14 = first 2 bits of axis status (AS) for axis 7
Bits 15-16 = first 2 bits of axis status (AS) for axis 8

The purpose of this command is to allow the user to create his or her own meaningful status word. It allows the user to place certain status information in the order they prefer.

The syntax `<B>INDUST<i><-<i><c>>` is described as follows:

- First `<i>` corresponds to the user status bit being defined (16 maximum).

- Second `<i>` corresponds to the bit of the axis status (AS), the system status (SS), the input status (IN), or the extended axis status (ASX).

- The `<c>` defines what status to use:

`<c>` Value	Function	`<c>` Value	Function
A	Use axis status (AS) for axis 1	K	RESERVED
B	Use axis status (AS) for axis 2	L	Use extended axis status (ASX) for axis 1
C	Use axis status (AS) for axis 3	M	Use extended axis status (ASX) for axis 2
D	Use axis status (AS) for axis 4	N	Use extended axis status (ASX) for axis 3
E	Use axis status (AS) for axis 5	O	Use extended axis status (ASX) for axis 4
F	Use axis status (AS) for axis 6	P	Use extended axis status (ASX) for axis 5
G	Use axis status (AS) for axis 7	Q	Use extended axis status (ASX) for axis 6
H	Use axis status (AS) for axis 8	R	Use extended axis status (ASX) for axis 7
I	Use system status (SS) *	S	Use extended axis status (ASX) for axis 8
J	Use input status (IN) **		

*   If you are using multitasking, the "I" value requires you to prefix the INDUST command with the task identifier (e.g., 2%INDUST6-2I assigns system status bit 2 for task 2 to user status bit 6). If no task prefix is given, the system status for task 1 is used by default.

**   The "J" value requires you to prefix the INDUST command with the I/O brick identifier (e.g., 2INDUST14-4J assigns the status of I/O point on I/O brick 2 to user status bit 14). If no brick prefix is given, the onboard trigger inputs are referenced by default. Refer to page 6 to fully understand the I/O bit patterns and use of the brick identifier.

Example
```
INDUSE1          ; Enable user status
INDUST1-5A       ; User status bit 1 defined as axis 1 status bit 5
INDUST2-3F       ; User status bit 2 defined as axis 6 status bit 3
3INDUST3-5J      ; User status bit 3 defined as input 5 on I/O brick 3
2%INDUST16-2I    ; User status bit 16 defined as system status bit 2 for task 2
```

INEN   Input Enable

Type	Input or Program Debug Tool	**Product**	**Rev**
Syntax	<!><B>INEN<d><d>...<d> (one <d> for each input)	6K	5.0
Units	n/a		
Range	d = 0 (disable, leave off), 1 (disable, leave on),		
	E (enable), or X (don't change)		
Default	E		
Response	INEN: *INENEEEE_EEEE_EEEE_EEEE_E		
	1INEN: *1INENEEEE_EEEE_EEEE_EEEE_EEEE_EEEE_EEEE_EEEE		
	1INEN.3 *E		
See Also	DRFEN, ERROR, [IN], INFNC, INLVL, INPLC, INSTW, LH, LIMEN,		
	TIN, TIO, TSTAT		

The INEN command allows you to simulate the activation of specific trigger or external digital inputs (without actually wiring the inputs to the controller) by disabling them and setting them to a specific level (ON or OFF).

The default INEN condition is enabled (E), requiring external wiring to exercise the input's respective INFNC function.

Using Inputs on Expansion I/O Bricks: If the I/O brick is disconnected or if it loses power, the controller will perform a kill (all tasks) and set error bit #18 (see ERROR). The controller will remember the brick configuration (volatile memory) in effect at the time the disconnection occurred. When you reconnect the I/O brick, the controller checks to see if anything changed (SIM by SIM) from the state when it was disconnected. If an existing SIM slot is changed (different SIM, vacant SIM slot, or jumper setting), the controller will set the SIM to factory default INEN and OUTLVL settings. If a new SIM is installed where there was none before, the new SIM is auto-configured to factory defaults.

Example: INEN1 disables trigger input A1 but leaves it in the ON state (the TIN command will show trigger input 1A as active). INEN∅ disables trigger input A1 but leaves it in the OFF (inactive) state. To re-enable trigger input 1A, issue the INENE command.

INEN **has no effect on …**

- trigger inputs when they are configured as "trigger interrupt" inputs with the INFNCi-H command. This includes position capture and registration functions.
- trigger or external digital inputs configured as "end-of-travel limit" inputs with the INFNCi-aR or INFNCi-aS commands. Instead, use the LH command.
- limit inputs found on your product's "LIMITS/HOME" connector(s).

Input bit assignments for the INEN command vary by product and external I/O brick configuration. The input bit patterns for onboard and external I/O bricks are explained on page 6 of this document.

Example:
```
DEF tester        ; Begin definition of program tester
WHILE(IN=b11X10)  ; While onboard inputs 1, 2, and 4 are active, and input 5 is not
                  ; active, execute the statements between the WHILE & NWHILE
GO1100            ; Initiate motion on axes 1 and 2
NWHILE            ; End WHILE statement
END               ; End definition of program tester
INEN11X10         ; Disable onboard inputs 1,2,4, & 5, and set inputs 1, 2 & 4 in
                  ; the active state, and input 5 in the inactive state
RUNtester         ; Initiate program tester
!INEN00000        ; Disable onboard inputs 1,2,3,4, & 5, and leave them in the
                  ; inactive state
INENeeeee         ; Re-enable inputs 1 through 5
```

INFNC   Input Function

			Product	**Rev**
Type	`Input`		6K	5.0
Syntax	`<!><B>INFNC<i>-<<a>c>`			
Units	`i = input #, a = axis #, c = function identifier letter`			

Range	`i = 1-32 (I/O brick dependent — see page 6);`
	`a = 1-8 (product dependent);`
	`c = A-T`			

Default	`A`			

Response	`INFNC:   (input function and status of onboard inputs)`
	`1INFNC:  (input function and status of I/O brick 1 inputs)`
	`1INFNC1: *1INFNC1-A NO FUNCTION - STATUS OFF`			

See Also	`COMEXR, COMEXS, ENCCNT, [ ER ], ERROR, [ IN ], INDEB, INEN,`
	`INLVL, INPLC, INSELP, INSTW, INTHW, JOY, JOYAXH, JOYAXL,`
	`JOYVH, JOYVL, K, KDRIVE, LH, LIMFNC, PSET, [ SS ], TER, TIN,`
	`TIO, TRGFN, TRGLOT, [ TRIG ], TSS, TSTAT, TTRIG`			

The Input Function (`INFNC`) command defines the function of each individual input, where `i` is the input bit number, `a` is an axis number if required, or the program number for the case of input function P, and `c` is the function. All function definitions given below will specify whether an axis number is required. A limit of 32 inputs may be assigned `INFNC` functions; this excludes functions A ("general-purpose") and H ("trigger interrupt").

Input Debounce. Using the Input Debounce Time (`INDEB`) command, you can change the input debounce time for all of the inputs on the specified I/O brick (all trigger inputs, found on the "TRIGGERS/OUTPUTS" connectors, are collectively considered I/O brick 0). The debounce is the period of time that the input must be held in a certain state before the controller recognizes it. This directly affects the rate at which the inputs can change state and be recognized. Trigger inputs that are assigned the "Trigger Interrupt" function (`INFNCi-H`), are instead debounced by the `TRGLOT` value. Inputs defined as limit inputs (`INFNCi-R`, `INFNCi-S`, or `INFNCi-T`), will not be debounced.

Input bit assignments vary by product. The input bit patterns for onboard and external I/O bricks are explained on page 6 of this document.

Input Scan Rate. The programmable inputs are scanned once per *system update* (2 milliseconds).

Multitasking. If the `INFNC` command does not include the task identifier (`%`) prefix, the function affects the task that executes the `INFNC` command. The functions that may be directed to a task with `%` are: C, D (without an axis specified), E, F, and P (e.g., `2%INFNC3-F` assigns onboard input 3 as a user fault input for task 2). Multiple tasks may share the same input, but the input may only be assigned one function.

Identifier	Function Description
A	**No special function** (general-purpose input). Normal input, used with the `IN` assignment
B	**BCD Program Select.** BCD input assignment to programs, lowest numbered input is least significant bit (LSB). BCD values for inputs are as follows:

	BCD Value
Least Significant Bit Value	1
.	2
.	4
.	8
.	10
.	20
.	40
.	80
Most Significant Bit Value	100

> **Note**: If fewer inputs than shown above are defined to be Program Select Inputs, then the highest input number defined as a Program Select Input is the most significant bit.

An input defined as a BCD Program Select Input will not function until the `INSELP` command has been enabled.

C   **Kill.** Kills motion on all axes and halts all command processing (refer to `K` and `KDRIVE` command descriptions for further details on the *kill* function). This is an edge detection function and is not intended to inhibit motion. To inhibit motion, use the Pause/Resume function (`INFNCi-E`). When enabled with the `ERROR` command, bit #6 of the `TER` and `ER` commands will report the kill status.

<a>D   **Stop.** Stops motion. Axis number is optional; if no axis number is specified, motion is stopped on all axes. If `COMEXS` is set to zero (`COMEXSØ`), program execution will be terminated. If `COMEXS` is set to 1 (`COMEXS1`), command processing will continue. With `COMEXS` set to 2 (`COMEXS2`), program execution is terminated, but the `INSELP` value is retained. Motion deceleration during the stop is controlled by the `AD` & `ADA` commands. If error bit #8 is enabled (e.g., `ERROR.8-1`), activating a Stop input will set the error bit and cause a branch to the `ERRORP` program.

E   **Pause/Continue.** If `COMEXR` is disabled (`COMEXRØ`), then only command execution pauses, not motion. With `COMEXR` enabled (`COMEXR1`), both command and motion execution are paused. After motion stops, you can release the input or issue a continue (`!C`) command to resume command processing (and motion of in `COMEXR1` mode).

F   **User Fault.** Refer to the `ERROR` command. If error bit #7 is enabled (e.g., `ERROR.7-1`), activating a User Fault input will set the error bit and cause a branch to the `ERRORP` program. **CAUTION**: Activating the user fault input sends an `!K` command to the controller, "killing" motion on all axes (refer to the `K` command description for ramifications).

G   **Reserved**

H   **Trigger Interrupt** - This function can only be assigned to the onboard trigger inputs. A "Trigger Interrupt" input can be used for these purposes:

- **Position Capture**. Each axis has two dedicated trigger inputs, referred to as "TRIG-nA" and "TRIG-nB" (n = number of the axis). These trigger inputs are located on the 25-pin "TRIGGERS/OUTPUTS" connector. When either trigger input (TRIG-nA or TRG-nB) for a particular axis is assigned the Trigger Interrupt function, activating the input performs a *hardware capture* of that axis' position. If the axis is used as a follower in Following, activating the trigger also performs an *interpolated capture* of the associated master axis position.

  An additional trigger, labeled "TRIG-M", may be used to perform a hardware capture of the "MASTER ENCODER" (the encoder connected to the "Master Encoder" connector), as well as the position of all axes (encoder position on servo axes; commanded or encoder position for steppers, depending on the `ENCCNT` setting). To assign TRIG-M as a trigger interrupt input, use the `INFNC17-H` command.

  When a Trigger Interrupt input is activated, the controller captures the relevant positions and stores them in registers that are available at the next system update (2 ms) through the use of these transfer and assignment/comparison commands:

Captured Information	Transfer	Assignment/Comparison	Offset *	Scale Factor **
Commanded position	`TPCC`	`PCC`	`PSET`	`SCLD`
Encoder position	`TPCE`	`PCE`	`PSET` or `PESET`	`SCLD`
Master encoder position	`TPCME`	`PCME`	`PMESET`	`SCLMAS`
Master cycle position	`TPCMS`	`PCMS`	`PSET`	`SCLMAS`

    *   Captured values are offset by any existing `PSET` or `PMESET` offset.
    ** If scaling is enabled, the captured position is scaled by `SCLD` or `SCLMAS`.

NOTES ABOUT POSITION CAPTURE:
- Hardware Capture: The encoder position is captured within ± 1 encoder count. The commanded position capture accuracy is ± 1 count.
- Interpolated Capture: There is a time delay of up to 50 µs between activating the trigger interrupt input and capturing the position; therefore, the accuracy of the captured position is equal to 50 µs multiplied by the velocity of the axis at the time the input was activated.
- Servo vs. Stepper. The nature of the axis position captured with a Trigger Interrupt input may be different, depending on whether the axis is configured for servo or stepper operation (`AXSDEF` command setting). For servo axes, both the commanded and encoder position for the axis are captured. Analog input feedback cannot be captured. For stepper axes, if the `ENCCNT` command is set to `ENCCNT0` (default condition), only the commanded position is captured. If `ENCCNT1` mode is enabled, only the encoder position is captured.

More about Trigger Interrupt function on next page …

H (con't.) *Continued from previous page (Trigger Interrupt function):*

- **Registration**. (see `RE` description for details)

- **Special trigger functions** defined with the `TRGFN` command (see `TRGFN` for details).

  NOTES ABOUT TRIGGER INTERRUPT INPUTS:
  - When a trigger is assigned the "Trigger Interrupt" function, the debounce is governed by the `TRGLOT` command setting (default is 24 ms). The `TRGLOT` setting overrides the existing `INDEB` setting for only the trigger inputs that are assigned the "Trigger Interrupt" function.
  - When configured as Trigger Interrupts, the triggers cannot be affected by the input enable (`INEN`) command.
  - Trigger Interrupt Status: Use the `TTRIG` and `TRIG` commands to ascertain if a trigger interrupt input has been activated. `TTRIG` displays the status as a binary report, and `TRIG` is an assignment/comparison operator for using the status information in a conditional expression (e.g., in an `IF` statement). The `TTRIG`/`TRIG` bits are cleared with the respective captured position is read (see table on previous page).

I **Alarm Event** - Will cause the 6K controller to set an Alarm Event in the Communications Server over the Ethernet interface. You must first enable the Alarm checking bit for this input-driven alarm (`INTHW.23-1`). For details on using alarms, refer to the *6K Series Programmer's Guide*.

aJ **JOG positive-direction** - Will jog the axis specified in a positive-direction. The `JOG` command must be enabled for this function to work. **Axis number required**.

aK **JOG negative-direction.** Will jog the axis specified in a negative-direction. The `JOG` command must be enabled for this function to work. **Axis number required**.

aL **JOG Speed Select.** Selects the high or low velocity range while jogging. If the input is active, the high jog velocity range will be selected. Axis number is optional. If no axis number is designated, it defaults to all axes.

M **Joystick Release.** Signals the controller to end joystick operation and resume program execution with the next statement in your program. When the input is open (high), the joystick mode is disabled (joystick mode can be enabled only if the input is closed, and only with the `JOY` command). When the input is closed (low), joystick mode can be enabled with the `JOY` command. The process of using Joystick mode is:

1. Assign the "Joystick Release" input function to a programmable input.
2. At the appropriate place in the program, enable joystick control of motion (with the `JOY` command). (Joystick mode cannot be enabled unless the "Joystick Release" input is closed.) When the `JOY` command enables joystick mode for the affect axes, program execution stops on those axes (assuming the Continuous Command Execution Mode is disabled with the `COMEXCØ` command).
3. Use the joystick to move the axes as required.
4. When you are finished using the joystick, open the "Joystick Release" input to disable the joystick mode. This allows program execution to resume with the next statement after the initial `JOY` command that started the joystick mode.

N **Joystick Axis Select.** Allows you to control two pairs of axes with one joystick. Use the `JOYAXH` and `JOYAXL` commands to assign analog inputs to control specific axes. Opening the Axis Select input (input is high) selects the `JOYAXH` configuration. Closing the Axis Select input (input is low) selects the `JOYAXL` configuration. NOTE: When this input is not connected, the `JOYAXH` configuration is always in effect.

O **Joystick Velocity Select.** Allows you to select the velocity for joystick motion. The `JOYVH` and `JOYVL` commands establish two joystick velocities. Opening the Velocity Select input (input is high) selects the `JOYVH` configuration. Closing the Velocity Select input (input is low) selects the `JOYVL` configuration. The `JOYVL` velocity could be used to quickly move to a location, the `JOYVH` velocity could be used for low-speed accurate positioning. NOTE: When this input is not connected, joystick motion always uses the `JOYVH` velocity setting.

P   **Program Select.** One to one correspondence for input vs. program number. The program number comes from the `TDIR` command. The number specified before the program name is the number to specify within this input definition. For example, in the `2INFNC1-3P` command, 3 is the program number. An input defined as a Program Select Input will not function until the `INSELP` command has been enabled.

Q   **Program Security.** Issuing the `INFNCi-Q` command enables the *Program Security* feature and assigns the *Program Access* function to the specified programmable input.

The program security feature denies you access to the `DEF`, `DEL`, `ERASE`, `MEMORY`, `LIMFNC`, and `INFNC` commands until you activate the program access input. Being denied access to these commands effectively restricts altering the user memory allocation. If you try to use these commands when program security is active (program access input is not activated), you will receive the error message `*ACCESS DENIED`. *The `INFNCi-Q` command is not saved in battery-backed RAM, so you may want to put it in the start-up program (`STARTP`).*

For example, once you issue the `3INFNC12-Q` command, the input on the 4th pin on SIM2 (I/O point 12) of I/O brick 3 is assigned the program access function and access to the `DEF`, `DEL`, `ERASE`, `MEMORY`, `LIMFNC`, and `INFNC` commands will be denied until you activate the input.

To regain access to these commands without the use of the program access input, you must issue the `INEN` command to disable the program security input, make the required user memory changes, and then issue the `INEN` command to re-enable the input. For example, if input 3 on brick 2 is assigned as the Program Security input, use `2INEN.3=1` to disable the input and leave it activated, make the necessary user memory changes, and then use `2INEN.3=E` to re-enable the input.

aR   **End-of-Travel Limit, Positive Direction.** This input function allows you to provide an end-of-travel limit input on your remove I/O brick. An axis number is required (e.g., `3INFNC1-4R` assigns the "Positive EOT limit" function to the 1st pin on the SIM1 (I/O point 1) on extended I/O brick #3, and makes it specific to axis 4). **REMEMBER** to reassign the corresponding dedicated hardware limit (on the "LIMITS/HOME" connector) to a function other than `LIMFNCi-aR`; otherwise, the `INFNCi-aR` input and the `LIMFNCi-aR` input will have the same function. Once an input is assigned a limit function, it is no longer debounced (`INDEB` has no effect), and it must be enabled/disabled with the `LH` command instead of the `INEN` command.

aS   **End-of-Travel Limit, Negative Direction.** This input function allows you to provide an end-of-travel limit input on your remove I/O brick. An axis number is required (e.g., `3INFNC2-4R` assigns the "Negative EOT limit" function to the 2nd pin on the SIM1 (I/O point 2) on extended I/O brick #3, and makes it specific to axis 4). **REMEMBER** to reassign the corresponding dedicated hardware limit (on the "LIMITS/HOME" connector) to a function other than `LIMFNCi-aS`; otherwise, the `INFNCi-aS` input and the `LIMFNCi-aS` input will have the same function. Once an input is assigned a limit function, it is no longer debounced (`INDEB` has no effect), and it must be enabled/disabled with the `LH` command instead of the `INEN` command.

aT   **Home Limit.** This input function allows you to provide a home limit input on your remote I/O brick. An axis number is required (e.g., `3INFNC3-4R` assigns the "Home limit" function to the 3rd pin on the SIM1 (I/O point 3) on extended I/O brick #3, and makes it specific to axis 4). **REMEMBER** to reassign the function of the home limit for the affected axis (e.g., given `3INFNC2-4T`, you must issue a `LIMFNC` command to assign a different function for the home input for axis 4). Once an input is assigned a limit function, it is no longer debounced (`INDEB` has no effect), and it must be enabled/disabled with the `LH` command instead of the `INEN` command.

Example:
```
3INFNC1-D       ; Input at I/O point #1 on brick 3 is defined to be a
                ; stop input for all axes
```

INLVL   Input Active Level

			Product	Rev
Type	`Input`		6K	5.0
Syntax	`<!><B>INLVL<b><b>...<b><b>`			
Units	`n/a`			
Range	`b = 0 (active low), 1 (active high), or X (don't change)`			
Default	`0`			
Response	`INLVL:     *INLVL0000_0000_0 (onboard trigger inputs)`			
	`1INLVL:    *1INLVL0000_0000_0000_0000_0000_0000_0000_0000`			
	`1INLVL.3: *0 (active low)`			
See Also	`INEN, INFNC, INPLC, INSTW, LIMLVL`			

The Input Active Level (`INLVL`) command defines the active state of all programmable inputs. To determine the input bit assignments for your 6K Series product, refer to page 6 of this document.

If the device driving the input is off (not sinking current), the input will show (using the `TIN` command) a zero (`0`) if the input has been defined as active low, and a one (`1`) if the input has been defined as active high. If the device driving the input is on (sinking current), the input will show a one (`1`) if the input has been defined as active low, and zero (`0`) if the input has been defined as active high. The default state is active low (`INLVL0`). The input schematics are provided in each 6K Series product's *Installation Guide*.

Example:
```
2INLVL0101       ; Set active level for these inputs on I/O brick 2:
                 ; inputs 1 & 3 are active low, inputs 2 & 4 are active high.
```

[INO]   Other Input Status

			Product	Rev
Type	`Assignment or Comparison`		6K	5.0
Syntax	`See below`			
Units	`n/a`			
Range	`n/a`			
Default	`n/a`			
Response	`n/a`			
See Also	`[ IN ], [ LIM ], TINO, TINOF`			

The Other Input Status (`INO`) command is used to assign an other input value 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, Ø, 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 Ø through 9.

Syntax: `VARBn=INO` where n is the binary variable number
        or `[ INO ]` can be used in an expression such as `IF(INO=b11Ø1)`, or `IF(INO=hØ2)`

There are 8 other inputs available for assignment or comparison. If it is desired to assign only one bit (one specific input) value to a binary variable, instead of all 8, use the bit select (`.`) operator . For example, `VARB1=INO.6` assigns the status of the ENABLE input to binary variable 1.

Format for binary assignment:
```
                   bbbbbbbb
                   ^      ^
                 Bit #1   Bit #8
```

Bit	Function	Location
1-5	RESERVED	
6	Enable input (`1` = OK for motion)	ENABLE terminal
7-8	RESERVED	

Example:
```
VARB2=INO.6      ; ENABLE input status assigned to binary variable 2
VARB2            ; Response if bit 6 is set to 1: *VARB2=XXXX_X1XX
IF(INO.6=b1)     ; If ENABLE input status is 1 (OK for motion), do the commands
                 ; following the  IF statement until the NIF statement
TREV             ; Transfer revision level
NIF              ; End if statement
```

INPLC   Establish PLC Data Inputs

		Product	**Rev**
Type	Input	6K	5.0
Syntax	`<!><B>INPLC<i>,<i-i>,<i>,<i>`		
Units	See below		
Range	See below		
Default	1,0—0,0,0		
Response	INPLC1: *INPLC1,0-0,0,0		
	1INPLC1: *1		
See Also	INEN, INFNC, INLVL, INSTW, OUTPLC, [TW]		

The Establish PLC Data Inputs (INPLC) command, in combination with the OUTPLC command, configure the inputs and outputs to read data from a parallel I/O device such as a PLC (Programmable Logic Controller), or a passive thumbwheel module. The actual data transfer occurs with the TW command. Refer to the TW command for a description of the data transfer process.

The INPLC command has four fields (`<i>,<i-i>,<i>,<i>`):

Data Field	Description
Field 1: `<i>`	**Set #**: There are 4 possible INPLC sets (1-4). This field identifies which set to use.
Field 2: `<i-i>`	**Input #s**: Data is read into the 6K Series product through the programmable inputs. This field identifies the inputs to be used with the TW command. The first number is the first input, and the second number is the last input. The inputs must be consecutive. The number of inputs should be 8, because two BCD digits are read per data strobe. Refer to page 6 for help in identifying which input bits are available to place in this field.
Field 3: `<i>`	**Sign Input #**: This field identifies which input is designated to provide sign information. A zero specified in the command field specifies no sign information. An active signal on the input designated as the sign input indicates a negative data entry.
Field 4: `<i>`	**Data Valid Input #**: This field identifies which input is designated to be the data valid handshake input. A zero in this field indicates that there will be no data valid handshake input used. When an input is specified as a data valid, the input must be active in order for data to be read. If the input is not active, data will not be read until the signal becomes active.

To disable a specific PLC set, enter `INPLCn,∅-∅,∅,∅` where n is the PLC set (1-4).

Example:
```
2INPLC2,1-8,9,10  ; Set INPLC set 2 as BCD digits on inputs 1-8 on I/O brick 2,
                  ; with input 9 as the sign bit, and input 10 as the data valid
1OUTPLC2,1-4,5,50 ; Set OUTPLC set 2 as output strobes on outputs 1-4 on I/O
                  ; brick 2, with output 5 as the output enable bit, and
                  ; strobe time of 50 milliseconds
A(TW6)            ; Read data into axis 1 acceleration using INPLC set 2 and
                  ; OUTPLC set 2 as the data configuration
```

www.comoso.com

INSELP
Select Program Enable

Type	Input	**Product**	**Rev**
Syntax	`<!>INSELP<i>,<i>`	6K	5.0
Units	See below		
Range	1st i = 0, 1, or 2; 2nd i = 0 – 5000		
Default	0,0		
Response	`INSELP: *INSELP0,0`		
See Also	COMEXS, INEN, INFNC, INLVL, INPLC, INSTW, LIMFNC, [SS], TDIR, TSS		

The Select Program Enable (`INSELP`) command enables program selection by inputs. In addition, the command establishes the strobe time for the inputs, and if programs are selected on a one-to-one basis (`INFNCi-iP` or `LIMFNCi-P`) or on a BCD basis (`INFNCi-B` or `LIMFNCi-B`). When programs are selected on a one-to-one basis, each input defined with the `INFNCi-iP` or `LIMFNCi-P` command will run a specific program upon activation. When programs are selected by BCD values, each input defined by the `INFNCi-B` or `LIMFNCi-B` command will contribute to the BCD value, which corresponds to the program number. The program number is derived from the order in which the programs were defined (`DEF`). The first program defined is program #1, the second defined is program #2, etc. To verify which program number corresponds to each program, use the `TDIR` command. The number in front of the program name is the program number.

First i =   Enable or disable function (∅ = Disable, 1 and 2 = Enable). Use `INFNCi-B` or `LIMFNCi-B` inputs if i = 1; use `INFNCi-iP` or `LIMFNCi-P` inputs if i = 2, to select program.

Second i =   Strobe Time in milliseconds for inputs used to select program. The input must be active at the end of the strobe time for it to be recognized as a valid selection. The inputs are scanned once per *system update* (2 milliseconds).

The Kill (`!K`) command releases this mode, in addition to `INSELP∅`. The Stop (`!S`) command or an input defined as a stop input will also release this mode, as long as `COMEXS` has been disabled.

Example:
```
2INFNC1-1P      ; Input #1 on I/O brick 2 defined to select program #1
2INFNC2-2P      ; Input #2 on I/O brick 2 defined to select program #2
2INFNC3-7P      ; Input #3 on I/O brick 2 defined to select program #7
INSELP2,50      ; Enable continuous scan of inputs to select a program to run
```

INSTW

Establish Thumbwheel Data Inputs

			Product	**Rev**
Type	`Input`		6K	5.0
Syntax	`<!><B>INSTW<i>,<i-i>,<i>`			
Units	See below			
Range	See below			
Default	`1,0—0,0`			
Response	`INSTW1: *INSTW1,0-0,0`			
	`1INSTW1: *1INSTW1,0-0,0`			
See Also	`INEN, INFNC, INLVL, INPLC, OUTTW, [ SS ], TSS, [ TW ]`			

The Establish Thumbwheel Data Inputs (`INSTW`) command, in combination with the `OUTTW` command, configure the inputs and outputs to read data from an active thumbwheel device. The actual data transfer occurs with the `TW` command. Refer to the `TW` command for a description of the data transfer process.

The `INSTW` command has three fields (`<i>,<i-i>,<i>`):

Data Field	Description
Field 1: `<i>`	**Set #**: There are 4 possible `INSTW` sets (1-4). This field identifies which set to use.
Field 2: `<i-i>`	**Input #s**: Data is read into the 6K Series product through the programmable inputs. This field identifies the inputs to be used with the `TW` command. The first number is the first input, and the second number is the last input. The inputs must be consecutive. The number of inputs should be compatible to the thumbwheel device. Refer to page 6 for help in identifying which input bits are available to place in this field.
Field 3: `<i>`	**Sign Input #**: This field identifies which input is designated to provide sign information. A zero specified in the command field specifies no sign information. An active signal on the input designated as the sign input indicates a negative data entry.

To disable a specific thumbwheel set, enter `INSTWn,Ø-Ø,Ø` where `n` is the thumbwheel set (1-4).

Example:
```
3INSTW2,1-4,5     ; Set INSTW set 2 as BCD digits on inputs 1-4 on I/O brick 3,
                  ; with input 5 as the sign bit
2OUTTW2,1-3,4,50  ; Set OUTTW set 2 as output strobes on outputs 1-3 on I/O
                  ; brick 2, with output 4 as the output enable bit, and
                  ; strobe time of 50 milliseconds
A(TW2)            ; Read data into axis 1 acceleration using INSTW set 2
                  ; and OUTTW set 2 as the data configuration
```

INTHW   Check for Alarm Events

			Product	**Rev**
Type	`Alarm Event`		6K	5.0
Syntax	`<!>INTHW<b><b>...<b><b>` (one b for each of 32 interrupts)			
Units	`n/a`			
Range	`b = 0 (disable), 1 (enable), or X (don't change)`			
Default	`0`			
Response	`INTHW:    *INTHW0000_0000_0000_0000_0000_0000_0000_0000`			
See Also	`INFNC, INTSW, LIMFNC, TIMINT`			

Use the `INTHW` command to determine which conditions will cause an alarm event in the 6K
Communications Server (this requires an Ethernet interface to the 6K). The alarm bit in the 6K is cleared as
soon as the alarm occurs, but the status of the alarm remains available, through the Communications Server,
to be checked by client applications. For details on using alarms, refer to the *6K Series Programmer's
Guide*.

The table below lists the potential alarm conditions, and any number of the conditions may be enabled.

Format for `INTHW`:   bbbb_bbbb_bbbb_bbbb_bbbb_bbbb_bbbb_bbbb
                                                     ^   ^
                                                 Bit #1   Bit #32
                                                Location   Location

To enable a specific interrupt, place a 1 in the corresponding bit location (`b`) in the `INTHWbb....bbb`
command. To disable a specific interrupt bit, place a ∅ in the corresponding bit location.

NOTE: A specific interrupt bit can also be enabled by specifying the bit and the state of the bit
(∅ = Disable, 1 = Enable). For example, the command `INTHW.29-1` enables bit 29, whereas
`INTHW.29-∅` disables bit 29.

Bit #	Function **	Bit #	Function
1	Software (forced) Alarm #1	17	Reserved
2	Software (forced) Alarm #2	18	Reserved
3	Software (forced) Alarm #3	19	Limit Hit - hard or soft limit, on any axis
4	Software (forced) Alarm #4	20	Stall Detected (stepper) or Position Error (servo) on any axis
5	Software (forced) Alarm #5	21	Timer (`TIMINT`)
6	Software (forced) Alarm #6	22	Reserved
7	Software (forced) Alarm #7	23	Input - any of the inputs defined by `INFNCi-I` or `LIMFNCi-I`
8	Software (forced) Alarm #8	24	Command Error
9	Software (forced) Alarm #9	25	Motion Complete on Axis 1
10	Software (forced) Alarm #10	26	Motion Complete on Axis 2
11	Software (forced) Alarm #11	27	Motion Complete on Axis 3
12	Software (forced) Alarm #12	28	Motion Complete on Axis 4
13	Command Buffer Full	29	Motion Complete on Axis 5
14	ENABLE input Activated	30	Motion Complete on Axis 6
15	Program Complete	31	Motion Complete on Axis 7
16	Drive Fault on any Axis	32	Motion Complete on Axis 8

** Bits 1-12: software alarms are forced with the `INTSW` command.

INTSW   Force an Alarm Event

Type	Alarm Event	**Product**	**Rev**
Syntax	`<!>INTSW<i>`	6K	5.0
Units	i = alarm event condition # (see list in INTHW)		
Range	i = 1-12		
Default	n/a		
Response	n/a		
See Also	INTHW		

This command forces a specific alarm event. The alarm events are available in the Communications Server (over the Ethernet interface), and a client application can read the Communications Server's "faster status" (alarm event) register to ascertain when certain conditions have occurred. 12 different software alarms are available (see table in `INTHW` command description). By forcing an alarm condition, you can customize the program to generate specific alarms at predefined places in your program.

The specific alarm event cannot be forced until the corresponding enable bit is set with the `INTHW` command. For example, before you can force alarm event bit #3 (`INTSW3`), you must first enable the 6K to check the state of alarm event bit #3 (`INTHW.3-1`).

The client application must determine the cause of the forced alarm event. This is accomplished by polling the Communication Server's "fast status" register for the alarm information. Once the register has been read for a client application, the alarm conditions are automatically cleared in the Communications Server. For more information on the alarms and using the fast status register, refer to the *Programmer's Guide*.

Example:
```
INTHW1             ; Enable alarm event bit #1
A20,20             ; Set acceleration to 20 units/sec/sec on axes 1 and 2
V2,2               ; Set velocity to 2 units/sec on axes 1 and 2
D25000,25000       ; Set move distance to 25000 units on axes 1 and 2
GO11               ; Initiate motion on axes 1 and 2
INTSW1             ; Force alarm event bit #1 as soon as the moves on axes 1 and
                   ; 2 are finished.
; **********************************************************************
; * Note: After the alarm occurs, it is the client application program's   *
; * responsibility to examine the communication server's fast status       *
; * register to determine the cause of the alarm.                          *
; **********************************************************************
```

JOG   Jog Mode Enable

Type	Jog		**Product**	**Rev**
Syntax	`<!><@><a>JOG<b><b><b><b><b><b><b>`		6K	5.0
Units	n/a			
Range	b = 0 (disable), 1 (enable), or X (don't change)			
Default	0			
Response	JOG: *JOG0000_0000			
	1JOG: *1JOG0			
See Also	DJOG, JOGA, JOGAA, JOGAD, JOGADA, JOGVH, JOGVL, INFNC, LIMFNC			

This command enables jog mode on the appropriate axis. Once jog mode has been enabled, the jog inputs can be used to produce motion on the specific axis. The inputs that will be used as jog inputs are determined by the INFNC or LIMFNC command. Once the jog inputs have been enabled, they will remain enabled, and able to jog at any time while the motor is *in position*. Or in other words, as long as the motor is not moving the jog inputs will be active.

After processing the JOG1 command, command processing does not stop and wait for the jog mode to be disabled (JOGØ). Instead, the jog inputs are enabled and command processing continues with the first command after the JOG1 command.

> **WARNING**
> If a jog input is active when jog mode is enabled, motion will occur.

To disable jog mode, issue the JOGØ command (to the appropriate axis) at any point in the program.

NOTE: If you are using an RP240 operator panel, you can enable the RP240 Jog Mode with the DJOG1 command and use the RP240's arrow keys to jog individual axes. To disable the RP240 Jog Mode, use the !DJOGØ command or press the RP240's **MENU RECALL** button.

Example:
```
1INFNC1-L        ; Input #1 on I/O brick 1 defined as jog velocity select input
1INFNC2-1J       ; Input #2 on I/O brick 1 defined as jog positive-direction
                 ; input for axis #1
1INFNC3-1K       ; Input #3 on I/O brick 1 defined as jog negative-direction
                 ; input for axis #1
1INFNC4-2J       ; Input #4 on I/O brick 1 defined as jog positive-direction
                 ; input for axis #2
1INFNC5-2K       ; Input #5 on I/O brick 1 defined as jog negative-direction
                 ; input for axis #2
JOGA100,100      ; Jog acceleration set to 100 units/sec/sec on both axes
JOGAD200,200     ; Jog deceleration set to 200 units/sec/sec on both axes
JOGVH10,8        ; The velocity when the jog velocity select input is high is
                 ; 10 units/sec on axis #1 and 8 units/sec on axis 2
JOGVL1,.8        ; The velocity when the jog velocity select input is low is
                 ; 1 units/sec on axis #1 and 0.8 units/sec on axis 2
JOG1100          ; Enable jog mode on axes 1 and 2. When an input occurs on
                 ; input 2, input 3, input 4, or input 5, the motor will move at
                 ; the appropriate jog velocity until the input is released
WAIT(1IN.6=b1)   ; Wait for input #6 on I/O brick 1 to become active.
                 ; Input #6 is being used as a signal to disable jog mode.
JOG0000          ; Disable jog mode on all axes
```

JOGA   Jog Acceleration

Type	Jog	**Product**	**Rev**
Syntax	`<!><@><a>JOGA<r>,<r>,<r>,<r>,<r>,<r>,<r>,<r>`	6K	5.0
Units	`r = units/sec/sec`		
Range	`0.00001 - 39,999,998 (depending on the scaling factor)`		
Default	`10.0000`		
Response	`JOGA:  *JOGA10.0000,10.0000,10.0000,10.0000 ...`		
	`1JOGA:  *1JOGA10.0000`		
See Also	`DJOG, JOG, JOGAA, JOGAD, JOGADA, JOGVH, JOGVL, INFNC, LIMFNC,`		
	`SCALE, SCLA`		

The Jog Acceleration (`JOGA`) command specifies the acceleration to be used upon receiving a jog input.

> **UNITS OF MEASURE** and **SCALING**: refer to page 16.

The jog acceleration remains set until you change it with a subsequent jog acceleration command. Accelerations outside the valid range are flagged as an error, with a message `*INVALID DATA-FIELD x`, where x is the field number. When an invalid acceleration is entered the previous acceleration value is retained.

If the jog deceleration (`JOGAD`) command has not been entered, the jog acceleration (`JOGA`) command will also set the jog deceleration rate. Once the jog deceleration (`JOGAD`) command has been entered, the jog acceleration (`JOGA`) command no longer affects jog deceleration.

Example: Refer to the jog mode enable (`JOG`) command example.

JOGAA   Jogging Average Acceleration

Type	Jog; Motion (S-Curve)	**Product**	**Rev**
Syntax	`<!><@><a>JOGAA<r>,<r>,<r>,<r>,<r>,<r>,<r>,<r>`	6K	5.0
Units	`r = units/sec/sec`		
Range	`0.00001 - 39,999,998 (depending on the scaling factor)`		
Default	`10.00 (trapezoidal profiling is default, where JOGAA tracks JOGA)`		
Response	`JOGAA:  *JOGAA10.0000,10.0000,10.0000,10.0000 ...`		
	`1JOGAA:  *1JOGAA10.0000`		
See Also	`A, ADA, JOG, JOGA, JOGAD, JOGADA, SCALE, SCLA`		

The Jogging Average Acceleration (`JOGAA`) command allows you to specify the average acceleration for an S-curve jogging profile. S-curve profiling provides smoother motion control by reducing the rate of change in acceleration and deceleration; this accel/decel rate of change is known as *jerk*. Refer to page 13 for details on S-curve profiling.

Scaling (`SCLA`) affects `JOGAA` the same as it does for `JOGA`. Refer to page 16 for details on scaling.

Example:
```
JOGA10,10,10,10    ; Sets the maximum jogging acceleration of all axes
JOGAA5,5,7.5,10    ; Sets the average jogging acceleration of all axes
```


JOGAD   Jog Deceleration

Type	Jog	**Product**	**Rev**
Syntax	`<!><@><a>JOGAD<r>,<r>,<r>,<r>,<r>,<r>,<r>,<r>`	6K	5.0
Units	`r = units/sec/sec`		
Range	`0.00001 - 39,999,998 (depending on the scaling factor)`		
Default	`10.0000 (JOGAD tracks JOGA)`		
Response	`JOGAD:  *JOGAD10.0000,10.0000,10.0000,10.0000 ...`		
	`1JOGAD:  *1JOGAD10.0000`		
See Also	`DJOG, JOG, JOGA, JOGAA, JOGADA, JOGVH, JOGVL, INFNC, LIMFNC,`		
	`SCALE, SCLA`		

The Jog Deceleration (`JOGAD`) command specifies the deceleration to be used when a jog input is released.

> **UNITS OF MEASURE** and **SCALING**: refer to page 16.

The jog deceleration remains set until you change it with a subsequent jog deceleration command. Decelerations outside the valid range are flagged as an error, with a message `*INVALID DATA-FIELD x`, where x is the field number. When an invalid deceleration is entered the previous deceleration value is retained.

If the jog deceleration (`JOGAD`) command has not been entered, the jog acceleration (`JOGA`) command will also set the jog deceleration rate. Once the jog deceleration (`JOGAD`) command has been entered, the jog acceleration (`JOGA`) command no longer affects jog deceleration. If `JOGAD` is set to zero (`JOGADØ`), then the jog deceleration will once again track whatever the `JOGA` command is set to.

Example: Refer to the jog mode enable (`JOG`) command example.

JOGADA   Jogging Average Deceleration

		Product	**Rev**
Type	`Jog; Motion (S-Curve)`	6K	5.0
Syntax	`<!><@><a>JOGADA<r>,<r>,<r>,<r>,<r>,<r>,<r>,<r>`		
Units	`r = units/sec/sec`		
Range	`0.00001 - 39,999,998 (depending on the scaling factor)`		
Default	`10.00 (JOGADA tracks JOGAA)`		
Response	`JOGADA: *JOGADA10.0000,10.0000,10.0000,10.0000 ...`		
	`1JOGADA: *1JOGADA10.0000`		
See Also	`A, AD, JOG, JOGA, JOGAA, JOGAD, SCALE, SCLA`		

The Jogging Average Deceleration (`JOGADA`) command allows you to specify the average deceleration for an S-curve jogging profile. S-curve profiling provides smoother motion control by reducing the rate of change in acceleration and deceleration; this accel/decel rate of change is known as *jerk*. Refer to page 13 for details on S-curve profiling.

Scaling (`SCLA`) affects `JOGADA` the same as it does for `JOGAD`. Refer to page 16 for details on scaling.

Example:
```
JOGAD10,10,10,10     ; Sets the maximum jog deceleration of all four axes
JOGADA5,5,7.5,10     ; Sets the average jog deceleration of all four axes
```

JOGVH   Jog Velocity High

		Product	**Rev**
Type	`Jog`	6K	5.0
Syntax	`<!><@><a>JOGVH<r>,<r>,<r>,<r>,<r>,<r>,<r>,<r>`		
Units	`r = units/sec (scalable with SCLV)`		
Range	`Stepper Axes: 0.00000-2,048,000 (max. depends on SCLV & PULSE)`		
	`Servo Axes:   0.00000-6,500,000 (max. depends on SCLV)`		
Default	`10.0000`		
Response	`JOGVH:  *JOGVH10.0000,10.0000,10.0000,10.0000 ...`		
	`1JOGVH:  *1JOGVH10.0000`		
See Also	`DJOG, JOG, JOGA, JOGAA, JOGAD, JOGADA, JOGVL, INFNC, LIMFNC,`		
	`PULSE, SCALE, SCLV`		

The Jog Velocity High (`JOGVH`) command specifies the velocity to be used upon receiving a jog input with the jog velocity select input active (*ON*).

The jog high velocity remains set until you change it with a subsequent jog high velocity command. Velocities outside the valid range are flagged as an error, with a message `*INVALID DATA-FIELD x`, where x is the field number. When an invalid velocity is entered the previous velocity value is retained.

UNITS OF MEASURE and **SCALING**: refer to page 16.

Example: Refer to the jog mode enable (`JOG`) command example.

JOGVL   Jog Velocity Low

		Product	**Rev**
Type	`Jog`	6K	5.0
Syntax	`<!><@><a>JOGVL<r>,<r>,<r>,<r>,<r>,<r>,<r>,<r>`		
Units	`r = units/sec (scalable with SCLV)`		
Range	Stepper Axes: 0.00000-2,048,000 (max. depends on SCLV & PULSE)		
	Servo Axes: 0.00000-6,500,000 (max. depends on SCLV)		
Default	0.5000		
Response	`JOGVL:    *JOGVL0.50000,0.50000,0.50000,0.50000 ...`		
	`1JOGVL:  *1JOGVL0.50000`		
See Also	DJOG, JOG, JOGA, JOGAA, JOGAD, JOGADA, JOGVH, INFNC, LIMFNC, PULSE, SCALE, SCLV		

The Jog Velocity Low (JOGVL) command specifies the velocity to be used upon receiving a jog input with the jog velocity select input low, or *OFF*. The velocity remains set until you change it with a subsequent jog velocity low command. Velocities outside the valid range are flagged as an error, with a message *INVALID DATA-FIELD x, where x is the field number. When an invalid velocity is entered the previous velocity value is retained.

> **UNITS OF MEASURE** and **SCALING**: refer to page 16.

Example: Refer to the jog mode enable (JOG) command example.

JOY   Joystick Mode Enable

		Product	**Rev**
Type	`Joystick`	6K	5.0
Syntax	`<!><@><a>JOY<b><b><b><b><b><b><b><b>`		
Units	n/a		
Range	b = 0 (disable), 1 (enable), or X (don't change)		
Default	0		
Response	`JOY:     *JOY0000_0000`		
	`1JOY:    *1JOY0`		
See Also	ANIRNG, [AS], COMEXC, INFNC, JOYA, JOYAA, JOYAD, JOYADA, JOYAXH, JOYAXL, JOYCDB, JOYCTR, JOYEDB, JOYVH, JOYVL, JOYZ, LIMFNC, TAS, TIN		

The 6K controller supports joystick operation with digital inputs and analog inputs. The digital inputs include the onboard limit inputs and trigger inputs, as well as digital input SIMs on an external I/O brick. The 12-bit analog inputs are available only if you install an analog input SIM on an external I/O brick (default voltage range is -10V to +10V, selectable with ANIRNG).

To Set Up Joystick Operation (refer also to the example code below):

1. Select the required digital inputs and analog inputs required for joystick operation. Connect the joystick as instructed in your controller's *Installation Guide*.

2. Assign the appropriate input functions to the digital inputs used for joystick's operation:
   - Release Input: INFNCi-M for triggers & external inputs, or LIMFNCi-M for limit inputs.
   - Axis Select Input: INFNCi-N for triggers & external inputs, or LIMFNCi-N for limit inputs. NOTE: If you're not using this input, assign the analog inputs to the axes with the JOYAXH command.
   - Velocity Select Input: INFNCi-O for triggers & external inputs, or LIMFNCi-O for limit inputs.

3. (optional) Use the ANIRNG command to select the voltage range for the analog inputs you will use. The default range is -10VDC to +10VDC (other options are 0 to +5V, -5 to +5V, and 0 to +10V).

4. Assign analog inputs to control specific axes, using:
   - JOYAXH: Standard analog input-to-axis assignment.
   - JOYAXL (underline): Analog input-to-axis assignment when the Axis Select Input is low.

5. Define the joystick motion parameters:
   - Max. Velocity when Velocity Select input switch is open/high (JOYVH command). If the Velocity Select input is not used, joystick motion always uses the JOYVH velocity.
   - Max. Velocity when Velocity Select input switch is closed/low (JOYVL command).
   - Accel (JOYA command).

- Accel for s-curve profiling (JOYAA command).
- Decel (JOYAD command).
- Decel for s-curve profiling (JOYADA command).

6. Define the usable voltage zone for your joystick:
   (make sure you have first assigned the analog inputs – see step 3 above)
   - End Deadband (JOYEDB): Defines the voltage offset (from the -10V & +10V endpoints) at which max. velocity occurs. Default is 0.1V, maxing voltage at -9.9V and +9.9V.
   - Center Voltage (JOYCTR or JOYZ): Defines the voltage when the joystick is at rest to be the zero-velocity center. Default JOYCTR setting is 0V.
   - Center Deadband (JOYCDB): Defines the zero-velocity range on either side of the Center Voltage. Default is 0.1V, setting the zero-velocity range at -0.1V to +0.1V.

7. To jog the axes:
   a. In your program, enable Joystick Operation with the JOY command (Joystick Release input must be closed in order to enable joystick mode). When the JOY command enables joystick mode for the affect axes, program execution stops on those axes (assuming the Continuous Command Execution Mode is disabled with the COMEXCØ command).
   b. Move the load with the joystick.
   c. When you are finished, open the Joystick Release input to disable joystick mode. This allows program execution to resume with the next statement after the initial JOY command that started the joystick mode.

Programming Example (refer also to the illustration below):

Application Requirements:
> This example represents a typical two-axis joystick application in which a high-velocity range is required to move to a region, then a low-velocity range is required for a fine search. After the search is completed it is necessary to record the load positions, then move to the next region. A digital input can be used to indicate that the position should be read. The Joystick Release input is used to exit the joystick mode and continue with the motion program.

Hardware Configuration:
> - An analog input SIM is installed in the 3rd slot of I/O brick 1. The eight analog inputs (1-8) are addressed as input numbers 17-24 on the I/O brick. Analog input 17 will control axis 1, and analog input 18 will control axis 2.
> - A digital input SIM is installed in the 1st slot of I/O brick 1. The eight digital inputs (1-8) are addressed as input numbers 1-8 on the I/O brick. Digital input 6 will be used for the Joystick Release function, and input 7 will be used for the Joystick Velocity Select input. Input 8 will be used to indicate that the position should be read.

Setup Code (the drawing below shows the usable voltage configuration):

```
1INFNC7-M           ; Assign Joystick Release f(n) to brick 1, input 7
1INFNC8-O           ; Assign Joystick Velocity Select f(n) to brick 1, input 8
JOYAXH1-17,1-18     ; Assign analog input 17 to control axis 1,
                    ; Assign analog input 18 to control axis 2
JOYVH1,1            ; Max. velocity on axes 1 & 2 is 10 units/sec when the
                    ; Velocity Select input switch (1IN.7) is open (high)
JOYVL10,10          ; Max. velocity on axes 1 & 2 is 1 unit/sec when the
                    ; Velocity Select input switch (1IN.7) is closed (low)
JOYA100,100         ; Set joystick accel to 100 units/sec/sec on both axes
JOYAD100,100        ; Set joystick decel to 100 units/sec/sec on both axes
;**** COMMANDS TO SET UP USABLE VOLTAGE: **********
1JOYCTR.17=+1.0     ; Set center voltage for analog input 17 (controls axis 1)
1JOYCTR.18=+1.0     ; and 18 (controls axis 2) to+1.0V. The +1.0V value was
                    ; ascertained by checking the voltage of the both
                    ; inputs (with the 1TANI.17 and 1TAIN.18 commands)
                    ; when the joystick was at rest.
1JOYCDB.17=0.5      ; Set center deadband to compensate for the fact that
1JOYCDB.18=0.5      ; when the joystick is at rest, the voltage received on
                    ; both analog inputs may fluctuate +/- 0.5V on either
                    ; side of the +1.0V center.
1JOYEDB.17=2.0      ; Set end deadband to compensate for the fact that the
1JOYEDB.18=2.0      ; joystick can produce only -8.0V to +8.0V.
;*************************************************
JOY11               ; Enable joystick mode for axes 1 & 2
```

Velocity
(positive direction)

JOYVH or JOYVL

JOYEDB

JOYCDB
(zero-velocity range)

Volts

-10V   +10V

JOYCTR or JOYZ
(voltage when joystick is at rest)

JOYEDB

JOYVH or JOYVL

Velocity
(negative direction)

JOYA   Joystick Acceleration

		Product	Rev
Type	`Joystick`	6K	5.0
Syntax	`<!><@><a>JOYA<r>,<r>,<r>,<r>,<r>,<r>,<r>,<r>`		
Units	`r = units/sec/sec`		
Range	`0.00120-39,999,998 (depending on the scaling factor)`		
Default	`10.0000`		
Response	`JOYA:    *JOYA10.0000,10.0000,10.0000,10.0000 ...`		
	`1JOYA:   *1JOYA10.0000`		
See Also	`JOY, JOYAA, JOYAD, JOYADA, JOYAXH, JOYAXL, JOYCDB, JOYCTR, JOYEDB,`		
	`JOYVH, JOYVL, JOYZ, SCALE, SCLA`		

The Joystick Acceleration (`JOYA`) command specifies the acceleration to be used during joystick mode.

> **UNITS OF MEASURE** and **SCALING**: refer to page 16.

The joystick acceleration remains set until you change it with a subsequent joystick acceleration command. Accelerations outside the valid range are flagged as an error, with a message `*INVALID DATA-FIELD x`, where x is the field number. When an invalid acceleration is entered the previous acceleration value is retained.

If the joystick deceleration (`JOYAD`) command has not been entered, the joystick acceleration (`JOYA`) command will also set the joystick deceleration rate. Once the joystick deceleration (`JOYAD`) command has been entered, the joystick acceleration (`JOYA`) command no longer affects joystick deceleration.

Example: Refer to the joystick mode enable (`JOY`) command example.

JOYAA   Joystick Average Acceleration

		Product	Rev
Type	`Motion (S-Curve)`	6K	5.0
Syntax	`<!><@><a>JOYAA<r>,<r>,<r>,<r>,<r>,<r>,<r>,<r>`		
Units	`r = units/sec/sec`		
Range	`0.00120-39,999,998 (depending on the scaling factor)`		
Default	`10.00 (trapezoidal profiling is default, where JOYAA tracks JOYA)`		
Response	`JOYAA:    *JOYAA10.0000,10.0000,10.0000,10.0000 ...`		
	`1JOYAA:   *1JOYAA10.0000`		
See Also	`AA, AD, JOY, JOYA, JOYAD, JOYADA, SCALE, SCLA`		

The Joystick Average Acceleration (`JOYAA`) command allows you to specify the average acceleration for an S-curve joystick profile. S-curve profiling provides smoother motion control by reducing the rate of change

---

in acceleration and deceleration; this accel/decel rate of change is known as *jerk*. Refer to page 13 for details on S-curve profiling.

Accelerating Scaling (SCLA) affects JOYAA the same as it does for JOYA. Refer to page 16 for details on scaling.

Example:
```
JOYA10,10,10,10    ; Set the maximum joystick acceleration of all four axes
JOYAA5,5,7.5,10    ; Set the average joystick acceleration of all four axes
```

JOYAD   **Joystick Deceleration**

Type	Joystick	**Product**	**Rev**
Syntax	<!><@><a>JOYAD<r>,<r>,<r>,<r>,<r>,<r>,<r>,<r>	6K	5.0
Units	r = units/sec/sec		
Range	0.00120-39,999,998 (depending on the scaling factor)		
Default	10.0000 (JOYAD tracks JOYA)		
Response	JOYAD: *JOYAD10.0000,10.0000,10.0000,10.0000 ...		
	1JOYAD: *1JOYAD10.0000		
See Also	JOY, JOYA, JOYAA, JOYADA, JOYAXH, JOYAXL, JOYCDB, JOYCTR,		
	JOYEDB, JOYVH, JOYVL, JOYZ, SCALE, SCLA		

The Joystick Deceleration (JOYAD) command specifies the deceleration to be used during the joystick mode.

> **UNITS OF MEASURE** and **SCALING**: refer to page 16.

The joystick deceleration remains set until you change it with a subsequent joystick deceleration command. Decelerations outside the valid range are flagged as an error, with a message *INVALID DATA-FIELD x, where x is the field number. When an invalid deceleration is entered the previous deceleration value is retained.

If the joystick deceleration (JOYAD) command has not been entered, the joystick acceleration (JOYA) command will also set the joystick deceleration rate. Once the joystick deceleration (JOYAD) command has been entered, the joystick acceleration (JOYA) command no longer affects joystick deceleration. If JOYAD is set to zero (JOYAD∅), then the joystick deceleration will once again track whatever the JOYA command is set to.

Example: Refer to the joystick mode enable (JOY) command example.

JOYADA   **Joystick Average Deceleration**

Type	Motion (S-Curve)	**Product**	**Rev**
Syntax	<!><@><a>JOYADA<r>,<r>,<r>,<r>,<r>,<r>,<r>,<r>	6K	5.0
Units	r = units/sec/sec		
Range	0.00120-39,999,998 (depending on the scaling factor)		
Default	10.00 (JOYADA tracks JOYAA)		
Response	JOYADA: *JOYADA10.0000,10.0000,10.0000,10.0000 ...		
	1JOYADA: *1JOYADA10.0000		
See Also	A, AD, JOY, JOYA, JOYAA, JOYAD, SCALE, SCLA		

The Joystick Average Deceleration (JOYADA) command allows you to specify the average deceleration for an S-curve joystick profile. S-curve profiling provides smoother motion control by reducing the rate of change in acceleration and deceleration; this accel/decel rate of change is known as *jerk*. Refer to page 13 for details on S-curve profiling.

Acceleration Scaling (SCLA) affects JOYADA the same as it does for JOYAD. Refer to page 16 for details on scaling.

Example:
```
JOYAD10,10,10,10    ; Sets the maximum joystick deceleration of all four axes
JOYADA5,5,7.5,10    ; Sets the average joystick deceleration of all four axes
```

JOYAXH   Joystick Analog Channel High

Type	Joystick		**Product**	**Rev**
Syntax	<!><@><a>JOYAXH<B-i>,<B-i>,<B-i>,<B-i>,<B-i>,<B-i>,<B-i>,<B-i>		6K	5.0
Units	B = I/O brick number			
	i = Location of the analog input on I/O brick (see page 6)			
Range	B = 1-8			
	i = 1-32			
Default	0-0,0-0,0-0,0-0,0-0,0-0,0-0,0-0			
Response	JOYAXH: *JOYAXH1-1,1-2,1-3,1-4,1-5,1-6,1-7,1-8			
	1JOYAXH: *1JOYAXH1-1			
See Also	ANIRNG, [IN], INFNC, JOY, JOYA, JOYAA, JOYAD, JOYADA, JOYAXL, JOYCDB, JOYCTR, JOYEDB, JOYVH, JOYVL, JOYZ, [LIM], LIMFNC, TIN, TLIM			

The Joystick Analog Channel High (JOYAXH) command specifies the analog input that will control each axis while the Joystick Axis Select input (INFNCi-N or LIMFNCi-N) is open and the corresponding axis is in Joystick Mode. A single analog input can control more than one axis (e.g., JOYAXH1-1,1-1 assigns the analog input at location 1 on I/O brick 1 to control axes 1 and 2). If the Joystick Axis Select input is not used, the JOYAXH command determines which axes are controlled by which analog inputs.

To understand how specific I/O points are addressed on the I/O bricks, refer to page 6.

NOTE: The 12-bit analog inputs are available only if you install an analog input SIM on an external I/O brick. Use the ANIRNG command to select the voltage range for the analog inputs you will use. The default range is -10VDC to +10VDC (other options are 0 to +5V, -5 to +5V, and 0 to +10V).

Example: Refer to the joystick mode enable (JOY) command example.

JOYAXL   Joystick Analog Channel Low

Type	Joystick		**Product**	**Rev**
Syntax	<!><@><a>JOYAXH<B-i>,<B-i>,<B-i>,<B-i>,<B-i>,<B-i>,<B-i>,<B-i>		6K	5.0
Units	B = I/O brick number			
	i = Location of the analog input on I/O brick (see page 6)			
Range	B = 1-8			
	i = 1-32			
Default	0-0,0-0,0-0,0-0,0-0,0-0,0-0,0-0			
Response	JOYAXL: *JOYAXL1-1,1-2,1-3,1-4,1-5,1-6,1-7,1-8			
	1JOYAXL: *1JOYAXL1-1			
See Also	ANIRNG,[IN], INFNC, JOY, JOYA, JOYAA, JOYAD, JOYADA, JOYAXH, JOYCDB, JOYCTR, JOYEDB, JOYVH, JOYVL, JOYZ, [LIM], LIMFNC, TIN, TLIM			

The Joystick Analog Channel Low (JOYAXL) command specifies the analog input that will control each axis while the Joystick Axis Select input (INFNCi-N or LIMFNCi-N) is closed and the corresponding axis is in Joystick Mode. A single analog input can control more than one axis (e.g., JOYAXL1-1,1-1 assigns the analog input at location 1 on I/O brick 1 to control axes 1 and 2). If the Joystick Axis Select input is not used, the JOYAXL command has no effect; instead, the JOYAXH command determines which axes are controlled by which analog inputs.

To understand how to address specific I/O points on the I/O bricks, refer to page 6.

NOTE: The 12-bit analog inputs are available only if you install an analog input SIM on an external I/O brick. Use the ANIRNG command to select the voltage range for the analog inputs you will use. The default range is -10VDC to +10VDC (other options are 0 to +5V, -5 to +5V, and 0 to +10V).

Example: Refer to the joystick mode enable (JOY) command example.

JOYCDB   **Joystick Center Deadband**

Type	Joystick		**Product**	**Rev**
Syntax	`<!><@><a><B>JOYCDB<.i><=r>`		6K	5.0
Units	`i = I/O location for the analog input on brick B (see page 6)`			
	`r = volts`			
Range	`i = 1-32`			
	`r = -10.00 - +10.00 (depending on ANIRNG setting)`			
Default	`0.1`			
Response	`1JOYCDB.1  *1JOYCDB.1=0.1`			
See Also	ANIRNG, INFNC, JOY, JOYA, JOYAA, JOYAD, JOYADA, JOYAXH, JOYAXL, JOYCTR, JOYEDB, JOYVH, JOYVL, JOYZ, LIMFNC			

The JOYCDB command defines, for the specified analog input(s), the zero-velocity range on either side of the Center Voltage established with the JOYCTR command or the JOYZ command. The default setting is 0.1V, which sets the zero-velocity range at -0.1V to +0.1V (assuming the default JOYCTR default of 0.0V is used). **NOTE**: Executing the JOYCDB command before the JOYAXH command will cause an error ("`INPUT(S) NOT DEFINED AS JOYSTICK INPUT`").

Example: Refer to the joystick mode enable (JOY) command example.

JOYCTR   **Joystick Center**

Type	Joystick		**Product**	**Rev**
Syntax	`<!><@><a><B>JOYCTR<.i><=r>`		6K	5.0
Units	`i = I/O location for the analog input on brick B (see page 6)`			
	`r = volts`			
Range	`i = 1-32`			
	`r = -10.00 - +10.00 (depending on ANIRNG setting)`			
Default	`0.00`			
Response	`1JOYCTR.1  *1JOYCTR.1=0.00`			
See Also	ANIRNG, INFNC, JOY, JOYA, JOYAA, JOYAD, JOYADA, JOYAXH, JOYAXL, JOYCDB, JOYEDB, JOYVH, JOYVL, JOYZ, LIMFNC			

The JOYCTR command defines, for the specified analog input(s), the voltage to be considered as the zero-velocity center (usually associated with leaving the joystick in the resting position). Default is 0V. The zero-velocity range about the center is determined by the JOYCDB command. As an alternative to the JOYCTR command, you could use the JOYZ command. **NOTE**: Executing the JOYCTR command before the JOYAXH command will cause an error ("`INPUT(S) NOT DEFINED AS JOYSTICK INPUT`").

Example: Refer to the joystick mode enable (JOY) command example.

JOYEDB   **Joystick End Deadband**

Type	Joystick		**Product**	**Rev**
Syntax	`<!><@><a><B>JOYEDB<.i><=r>`		6K	5.0
Units	`i = I/O location for the analog input on brick B (see page 6)`			
	`r = volts`			
Range	`i = 1-32`			
	`r = -10.00 - +10.00 (depending on ANIRNG setting)`			
Default	`0.1`			
Response	`1JOYEDB.1  *1JOYCTR.1=0.1`			
See Also	ANIRNG, INFNC, JOY, JOYA, JOYAA, JOYAD, JOYADA, JOYAXH, JOYAXL, JOYCDB, JOYCTR, JOYVH, JOYVL, JOYZ, LIMFNC			

The JOYEDB command defines, for the specified analog input(s), the voltage offset (from the -10V & +10V endpoints) at which maximum velocity occurs. This command is useful if your joystick does not reach either limit of the voltage range (-10.00V to+10.00V). The default setting is 0.1V, creating a maximum voltage range of -9.9V to +9.9V. **NOTE**: Executing the JOYEDB command before the JOYAXH command will cause an error ("`INPUT(S) NOT DEFINED AS JOYSTICK INPUT`").

Example: Refer to the joystick mode enable (JOY) command example.

JOYVH   Joystick Velocity — Velocity Select Input High

		Product	**Rev**
Type	`Joystick`		
Syntax	`<!><@><a>JOYVH<r>,<r>,<r>,<r>,<r>,<r>,<r>,<r>`	6K	5.0
Units	`r = units/sec`		
Range	`Stepper Axes: 0.00000-2,048,000 (max. depends on SCLV & PULSE)`		
	`Servo Axes:   0.00000-6,500,000 (max. depends on SCLV)`		
Default	`0.5000`		
Response	`JOYVH:   *JOYVH0.5000,0.5000,0.5000,0.5000 ...`		
	`1JOYVH:  *1JOYVH0.5000`		
See Also	`[ IN ], INFNC, JOY, JOYA, JOYAA, JOYAD, JOYADA, JOYAXH, JOYAXL,`		
	`JOYCDB, JOYCTR, JOYEDB, JOYVL, JOYZ, [ LIM ], LIMFNC, PULSE.`		
	`SCALE, SCLV, TIN, TLIM`		

The Joystick Velocity High (`JOYVH`) command specifies the maximum velocity that can be obtained at full deflection during joystick mode, with the Joystick Velocity Select input open (high). The Joystick Velocity Select input function is defined with the `INFNCi-O` command or the `LIMFNCi-O` command. If the Velocity Select input is not used, joystick motion always uses the `JOYVH` velocity.

NOTE: The data fields (`<r>,<r>,<r>,<r>...`) represent the axes, **not the analog inputs**.

The joystick velocity must be entered prior to entering joystick mode (`JOY`). The joystick velocity high remains set until you change it with a subsequent `JOYVH` command. Velocities outside the valid range are flagged as an error, with a message `*INVALID DATA-FIELD x`, where `x` is the field number. When an invalid velocity is entered the previous velocity value is retained.

> **UNITS OF MEASURE** and **SCALING**: refer to page 16.

Example: Refer to the joystick mode enable (`JOY`) command example.

JOYVL   Joystick Velocity — Velocity Select Input Low

		Product	**Rev**
Type	`Joystick`		
Syntax	`<!><@><a>JOYVL<r>,<r>,<r>,<r>,<r>,<r>,<r>,<r>`	6K	5.0
Units	`r = units/sec`		
Range	`Stepper Axes: 0.00000-2,048,000 (max. depends on SCLV & PULSE)`		
	`Servo Axes:   0.00000-6,500,000 (max. depends on SCLV)`		
Default	`0.2000`		
Response	`JOYVL:   *JOYVL0.2000,0.2000,0.2000,0.2000 ...`		
	`1JOYVL:  *1JOYVL0.2000`		
See Also	`[ IN ], INFNC, JOY, JOYA, JOYAA, JOYAD, JOYADA, JOYAXH, JOYAXL,`		
	`JOYCDB, JOYCTR, JOYEDB, JOYVH, JOYZ, [ LIM ], LIMFNC, PULSE.`		
	`SCALE, SCLV, TIN, TLIM`		

The Joystick Velocity Low (`JOYVL`) command specifies the maximum velocity that can be obtained at full deflection during joystick mode, with the Joystick Velocity Select input closed (low). The Joystick Velocity Select input function is defined with the `INFNCi-O` command or the `LIMFNCi-O` command. If the Velocity Select input is not used, joystick motion always uses the `JOYVH` velocity.

NOTE: The data fields (`<r>,<r>,<r>,<r>...`) represent the axes, **not the analog channels**.

The joystick velocity must be entered prior to entering joystick mode (`JOY`). The joystick velocity low remains set until you change it with a subsequent `JOYVL` command. Velocities outside the valid range are flagged as an error, with a message `*INVALID DATA-FIELD x`, where `x` is the field number. When an invalid velocity is entered the previous velocity value is retained.

> **UNITS OF MEASURE** and **SCALING**: refer to page 16.

Example: Refer to the joystick mode enable (`JOY`) command example.

JOYZ   **Joystick Zero**

Type	Joystick		**Product**	**Rev**
Syntax	`<!><@><B>JOYZ<.i><=b>`		6K	5.0
	(multiple inputs per brick may be configured at one time)			
Units	B = I/O brick number			
	i = Location of the analog input on I/O brick (see page 6)			
	b = enable bit			
Range	B = 1-8			
	i = 1-32			
	b = 0 (don't zero), 1 (zero), or X (don't change)			
Default	n/a			
Response	n/a			
See Also	ANIRNG, JOY, JOYA, JOYAA, JOYAD, JOYADA, JOYAXH, JOYAXL,			
	JOYCDB, JOYCTR, JOYEDB, JOYVH, JOYVL			

The Joystick Zero (JOYZ) command defines the voltage when the joystick is at rest to be the zero-velocity center. Simply leave the joystick in its resting position and issue a JOYZ command to define the current voltage of the respective analog inputs as the zero-velocity center. The zero-velocity range about the center is determined by the JOYCDB command.

The JOYZ command is an alternative to using the JOYCTR command.

Example:
```
1INFNC7-M          ; Assign Joystick Release f(n) to brick 1, input 7
1INFNC8-O          ; Assign Joystick Velocity Select f(n) to brick 1, input 8
JOYAXH1-17,1-18    ; Assign analog input 17 to control axis 1,
                   ; Assign analog input 18 to control axis 2
JOYVH10,10         ; Max. velocity on axes 1 & 2 is 10 units/sec when the
                   ; Velocity Select input (1IN.7) is high (sinking current)
JOYVL1,1           ; Max. velocity on axes 1 & 2 is 1 unit/sec when the
                   ; Velocity Select input (1IN.7) is low (not sinking current)
JOYA100,100        ; Set joystick accel to 100 units/sec/sec on both axes
JOYAD100,100       ; Set joystick decel to 100 units/sec/sec on both axes
;**** COMMANDS TO SET UP USABLE VOLTAGE: **********
1JOYZ.17=1         ; These command are executed while the joystick is at rest. They
1JOYZ.18=1         ; set the current voltage on analog input 17 (controls axis 1)
                   ; and input 18 (controls axis 2) as the zero-velocity center.
1JOYCDB.17=0.5     ; Set center deadband to compensate for the fact that
1JOYCDB.18=0.5     ; when the joystick is at rest, the voltage received on
                   ; both analog inputs may fluctuate +/- 0.5V on either
                   ; side of the zero-velocity center established with JOYZ.
1JOYEDB.17=2.0     ; Set end deadband to compensate for the fact that the
1JOYEDB.18=2.0     ; joystick can produce only -8.0V to +8.0V.
;***********************************************
JOY11              ; Enable joystick mode for axes 1 & 2
```

JUMP   **Jump to a Program or Label (and do not return)**

Type	Program or Subroutine Definition or Program Flow Control		**Product**	**Rev**
Syntax	`<!>JUMP<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, GOTO, IF, L, LN, NIF, NWHILE, REPEAT,			
	RUN, UNTIL, WHILE			

The JUMP command branches to the corresponding program name or label when executed. A program or label name consists of 6 or fewer alpha-numeric characters.

All nested IFs, WHILEs, and REPEATs, loops, and subroutines are cleared; thus, the program or label that the JUMP initiates will **not** return control to the line after the JUMP, when the program completes operation. Instead, the program will end.

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

```
; ***********************************************************************
; * In this example, the program place is executed and calls the pick  *
; * subroutine. The pick subroutine then initiates motion on axes 1 & 2 *
; * (GO1100) and jumps to the program called load to initiate motion on *
; * axis 3 (GO001). Then, because the JUMP command cleared the pick      *
; * subroutine, program execution is terminated instead of returning to *
; * the place program.                                                   *
; ***********************************************************************
DEF pick           ; Begin definition of subroutine named pick
GO1100             ; Initiate motion on axes 1 and 2
JUMP load          ; Jump to the program named load
END                ; End subroutine definition
DEF load           ; Begin definition of program named load
GO001              ; Initiate motion on axis 3
END                ; End program 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
RUN place          ; Execute program named place
```

K   Kill Motion

Type	Motion		**Product**	**Rev**
Syntax	`<!><@>K<b><b><b><b><b><b><b>`		6K	5.0
Units	n/a			
Range	b = 0 (don't kill), 1 (kill), or X (don't change)			
Default	n/a			
Response	!K No response, instead motion is killed on all axes			
See Also	DRFLVL, FOLK, GO, <CTRL>K, KDRIVE, LHAD, LHADA, S, SCANP, TAS			

The Kill Motion (K) command instructs the motor to stop motion on the specified axes. If the Kill (K) command is used without any arguments (K or !K), motion will be stopped on all axes, and program execution will be terminated. When the Kill (K) command is used with ones in the command fields (e.g., KØ11Ø), motion will be stopped on the axes specified with ones (1), and program execution will continue with the next command. The Kill command will be used most frequently with the immediate command delimiter in front of the command (!K). By using the immediate Kill (!K) command, motion will be stopped at the time the command is received.

Motion is stopped at the rate set with the LHADA and LHAD commands. If you want the drive to be disabled upon executing a K or !K command, enable the *Disable Dive on Kill* mode with the KDRIVE command. **CAUTION**: In the KDRIVE mode, a K or !K command immediately shuts down the drive, allowing the load to *free wheel* to a stop.

If the axis is involved in a PLC Scan (initiated with SCANP), a K command will clear the scan.

Example:
```
A2,2,25000,25000   ; Set acceleration to 2, 2, 25000, and 25000 units/sec/sec
                   ; for axes 1, 2, 3 and 4
AD2,2,25000,25000  ; Set deceleration to 2, 2, 25000, and 25000 units/sec/sec
                   ; for axes 1, 2, 3 and 4
V1,1,1,2           ; Set velocity to 1, 1, 1, and 2 units/sec for axes 1, 2, 3
                   ; and 4, respectively
@D10               ; Set distance on all axes to 10 units
@GO1               ; Initiate motion on all axes -- motion begins.
                   ; After a short period the Kill command is sent.
!K                 ; Kill motion on all axes (steppers stop instantaneously,
                   ; servos stop at the LHADA/LHAD decel)
```

\<CTRL>K Kill Motion

			Product	**Rev**
Type	`Motion`		6K	5.0
Syntax	`<CTRL>K`			
Units	`n/a`			
Range	`n/a`			
Default	`n/a`			
Response	`<CTRL>K: No response, instead motion is killed on all axes`			
See Also	`GO, K, KDRIVE, LHAD, LHADA, S`			

The Kill Motion (`<ctrl>K`) command instructs the controller to stop motion on all axes, and terminate program execution. In essence, the `<ctrl>K` command is an immediate kill (`!K`) command.

Motion is stopped at the rate set with the `LHADA` and `LHAD` commands. If the *Disable Dive on Kill* mode is enabled with the `KDRIVE` command, a `<ctrl>K` command immediately shuts down the drive, allowing the load to *free wheel* to a stop.

Example:
```
A2,2,25000,25000   ; Sets acceleration to 2, 2, 25000, and 25000 units/sec/sec
                   ; for axes 1, 2, 3 and 4
AD2,2,25000,25000  ; Sets deceleration to 2, 2, 25000, and 25000 units/sec/sec
                   ; for axes 1, 2, 3 and 4
V1,1,1,2           ; Sets velocity to 1, 1, 1, and 2 units/sec for axes 1, 2, 3
                   ; & 4, respectively
@D10               ; Set distance on all axes to 10 units
@GO1               ; Initiate motion on all axes -- motion begins.
                   ; After a short period the Kill command is sent.
<CTRL>K            ; Kill motion on all axes (steppers stop instantaneously,
                   ; servos stop at the LHADA/LHAD deceleration)
```

KDRIVE Disable Drive on Kill

			Product	**Rev**
Type	`Controller Configuration`		6K	5.0
Syntax	`<!><@><a>KDRIVE<b><b><b><b><b><b><b><b>`			
Units	`b = enable bit`		(applicable to servo axes only)	
Range	`0 (disable), 1 (enable), or X (don't change)`			
Default	`0`			
Response	`KDRIVE:  *KDRIVE0000_0000`			
	`1KDRIVE: *KDRIVE0`			
See Also	`DRIVE, INFNC, K, <ctrl>K, LIMFNC`			

If you enable the Disable Drive on Kill function (`KDRIVE1`), then when a kill command (`K`, `!K`, or `<ctrl>K`) is processed or a kill input (`INFNCi-C` or `LIMFNCi-C`) is activated, the drive will be disabled immediately; this cuts all control to the motor and allows the load to freewheel to a stop (although steppers have some detent torque).

When the drive is disabled (shutdown/de-energized):

- Stepper Axis: Shutdown+ sources current and Shutdown- sinks current.
- Servo Axis: SHTNO relay output is disconnected from COM, and the SHTNC relay output is connected to COM.

To re-enable the drive, issue the `DRIVE1` command to the affect axis or axes.

If you leave the `KDRIVE` command in its default state (∅, disabled), the kill function behaves in its normal manner, leaving the drive enabled.

Example:
```
KDRIVE11           ; Set axes 1 & 2 to de-energize the drive during a kill
K                  ; Kill is performed and drives are de-energized
```

L   Loop

Type	Loops; Program Flow Control	**Product**	**Rev**
Syntax	`<!>L<i>`	6K	5.0
Units	i = number of times to loop		
Range	0-999,999,999		
Default	0		
Response	L: No response; instead, this has the same function as L0		
See Also	LN, LX, PLN, PLOOP		

When you combine the Loop (`L`) command with the end of loop (`LN`) command, all of the commands between `L` and `LN` will be repeated the number of times indicated by n. If `<i>` = ∅, or if no argument is specified, all the commands between `L` and `LN` will be repeated indefinitely. The loop can be stopped by issuing a Terminate Loop (`!LX`) command, an immediate Kill (`!K`) command, or an immediate Halt (`!HALT`) command.

The loop can be paused by issuing an immediate Pause (`!PS`) command or a Stop (`!S`) command with `COMEXS` enabled. The loop can then be resumed with the immediate Continue (`!C`) command. You may nest loops up to 16 levels deep.

NOTE: Be careful about performing a `GOTO` between the `L` and `LN` commands. Branching to a different location within the same program will cause the next loop encountered to be nested within the previous loop, unless an `LN` command has already been encountered.

Example:
```
L5          ; Repeat the commands between L and LN five times
GO1110      ; Start motion on axes 1, 2, and 3, axis 4 will remain motionless
LN          ; End loop
```

LH   Hardware End-of-Travel Limit — Enable Checking

Type	Limit(End-of-Travel)	**Product**	**Rev**
Syntax	`<!><@><a>LH<i>,<i>,<i>,<i>,<i>,<i>,<i>`	6K	5.0
Units	n/a		
Range	i = 0 (disable both), 1 (disable positive-direction), 2 (disable negative-direction), or 3 (enable both)		
Default	3		
Response	LH: *LH3,3,3,3,3,3,3,3 1LH: *1LH3		
See Also	[AS], [ER], ERROR, INFNC, INLVL, LHAD, LHADA, [LIM], LIMEN, LIMFNC, LIMLVL, LS, LSAD, LSADA, LSNEG, LSPOS, TAS, TER, TLIM, TSTAT		

Use the `LH` command to enable or disable the inputs defined as end-of-travel limit inputs. This pertains to onboard limit inputs defined with the `LIMFNCi-aR` and `LIMFNCi-aS` commands (this is the factory default configuration for limits), as well as to onboard triggers and external digital inputs defined with the `INFNCi-aR` and `INFNCi-aS` commands.

Command Syntax:

```
        LH<i>,<i>,<i>,<i>,<i>,<i>,<i>,<i>
Axis 1 ─────────┘ │ │ │
Axis 2 ───────────┘ │ │
Axis 3 ─────────────┘ │
Axis 4 ───────────────┘
```
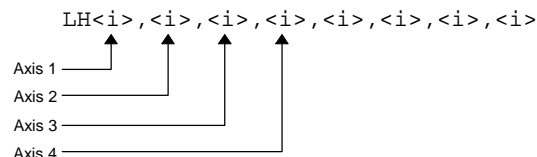
With limits disabled, motion will not be restricted. When a specific limit is enabled (positive- or negative-direction), and the limit wiring for the enabled limit is a physical open circuit, motion will be restricted (assuming `LHLVL∅` or `INLVL∅`). The `LHLVL` controls the active level for onboard limit inputs, and the `INLVL` command controls the active level for onboard triggers and external digital inputs.

Disable negative-direction limit; Disable positive-direction limit:	i = 0
Enable negative-direction limit; Disable positive-direction limit:	i = 1
Disable negative-direction limit; Enable positive-direction limit:	i = 2
Enable negative-direction limit; Enable positive-direction limit:	i = 3

www.comoso.com

If an "end-of-travel limit" input is redefined with a different function (i.e., not LIMFNCi-R, LIMFNCi-S, INFNCi-R or INFNCi-S), it is no longer controlled by the LH command. If the input is a limit (on the "LIMITS/HOME" connector), use the LIMEN command; if the input is a trigger or external digital input, use the INEN command.

NOTE

If a hard limit is encountered while limits are enabled, motion must occur in the opposite direction after correcting the limit condition (resetting the switch); then you can make a move in the original direction. If limits are disabled, you are free to make a move in either direction.

Example:
```
LH3,3              ; Enable limits on axes 1 and 2
LHAD100,100        ; Set hard limit decel to 100 units/sec/sec on axes 1 and 2
LIMLVL00x00        ; Active low hard limits for axes 1 & 2
A10,12             ; Set acceleration to 10 and 12 units/sec/sec for axes 1 and 2
V1,1               ; Set velocity to 1 unit/sec for axes 1 and 2
D100000,1000       ; Set distance to 100000 and 1000 units for axes 1 and 2
GO11XX             ; Initiate motion on axes 1 and 2
```

LHAD   Hard Limit Deceleration

		Product	Rev
Type	Limit (End-of-Travel)	**6K**	**5.0**
Syntax	<!><@><a>LHAD<r>,<r>,<r>,<r>,<r>,<r>,<r>,<r>		
Units	r = units/sec/sec		
Range	0.00001-39,999,998 (depending on the scaling factor)		
Default	100.0000		
Response	LHAD: *LHAD100.0000,100.0000,100.0000,100.0000 ...		
	1LHAD: *1LHAD100.0000		
See Also	DRES, DRFLVL, INFNC, K, LH, LHADA, [LIM], LIMFNC, LIMLVL, LS, LSAD, LSADA, LSNEG, LSPOS, SCALE, SCLA		

The Hard Limit Deceleration (LHAD) command determines the value at which to decelerate after an end-of-travel limit has been hit. This applies to the on-board dedicated limits, as well as to any inputs configured as end-of-travel limits (INFNCi-R or INFNCi-S).

UNITS OF MEASURE and **SCALING**: refer to page 16.

When a drive fault, a Kill command (K, !K, or ^K), or a Kill input (INFNCi-C or LIMFNCi-C) occurs, motion is stopped at the rate set with the LHAD and LHADA commands. If the *Disable Drive on Kill* mode is enabled (KDRIVE1), the drive is immediately shut down upon a Kill command or input and allows the motor/load to *freewheel* to a stop without a controlled deceleration.

The hard limit deceleration remains set until you change it with a subsequent hard limit deceleration command. Decelerations outside the valid range are flagged as an error, with a message *INVALID DATA-FIELD x, where x is the field number. When an invalid deceleration is entered the previous deceleration value is retained.

Example: Refer to the hard limit enable (LH) command example.

LHADA   Hard Limit Average Deceleration

		Product	Rev
Type	Motion (S-Curve)	**6K**	**5.0**
Syntax	<!><@><a>LHADA<r>,<r>,<r>,<r>,<r>,<r>,<r>,<r>		
Units	r = units/sec/sec		
Range	0.00001-39,999,998 (depending on the scaling factor)		
Default	100.000 (default is a constant deceleration ramp, where LHADA tracks LHAD)		
Response	LHADA: *LHADA100.0000,100.000,100.000,100.000 ...		
	1LHADA: *1LHADA100.0000		
See Also	AD, ADA, INFNC, K, LHAD, LIMFNC, LIMLVL, SCALE, SCLA		

The Hard Limit Average Deceleration (LHADA) command allows you to specify the average deceleration for an S-curve deceleration profile when a limit is hit. S-curve profiling provides smoother motion control by

reducing the rate of change in deceleration; this decel rate of change is known as *jerk*. Refer to page 13 for details on S-curve profiling.

Acceleration scaling (`SCLA`) affects `LHADA` the same as it does for `LHAD`. Refer to page 16 for details on scaling.

Example:
```
LHAD10,10,10,10    ; Set the maximum deceleration of axes 1-4
LHADA5,5,7.5,10    ; Set the average deceleration of axes 1-4
```

[LIM]   **Limit 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	[IN], INDEB, INFNC, LH, LIMEN, LIMFNC, LIMLVL, TLIM			

The Limit Status (`LIM`) operator is used to assign the limit 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, Ø, 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 Ø through 9.

`LIM` <u>does not</u> depict the status of trigger inputs or external digital inputs assigned an end-of-travel or home limit function (`INFNCi-R`, `INFNCi-S`, or `INFNCi-T`). For such inputs, you must use the `IN` operator.

Syntax: `VARBn=LIM` where n is the binary variable number,
or `LIM` can be used in an expression such as `IF(LIM=b1XX1)`, or `IF(LIM=h7)`

The `LIM` value is the debounced version of the limits status (debounced with the `ØINDEB` value). Axis status (`AS`) bits 15 and 16 reports the non-debounced version of the end-of-travel limits.

There are 3 limit inputs per axis, home limit, positive-direction, and negative-direction end-of-travel limits. Each is available for assignment or comparison. If it is desired to assign only one limit input value to a binary variable, instead of the status of all the limit inputs, the bit select (`.`) operator can be used. The bit select, in conjunction with the limit input number, is used to specify a specific limit input. For example, `VARB1=LIM.4` assigns limit input 4 (positive-direction limit for axis 2) to binary variable 1.

Format for binary assignment: bbbbbbbbbbbbbbbbbbbbbbbb
                                 ^   ^
                              Bit #1   Bit #24

`LIM` **bit**	**Function**	`LIM` **bit**	**Function**
1	Axis 1 - positive-direction Limit	13	Axis 5 - positive-direction Limit
2	Axis 1 - negative-direction Limit	14	Axis 5 - negative-direction Limit
3	Axis 1 - Home Limit	15	Axis 5 - Home Limit
4	Axis 2 - positive-direction Limit	16	Axis 6 - positive-direction Limit
5	Axis 2 - negative-direction Limit	17	Axis 6 - negative-direction Limit
6	Axis 2 - Home Limit	18	Axis 6 - Home Limit
7	Axis 3 - Positive-direction Limit	19	Axis 7 - Positive-direction Limit
8	Axis 3 - Negative-direction Limit	20	Axis 7 - Negative-direction Limit
9	Axis 3 - Home Limit	21	Axis 7 - Home Limit
10	Axis 4 - Positive-direction Limit	22	Axis 8 - Positive-direction Limit
11	Axis 4 - Negative-direction Limit	23	Axis 8 - Negative-direction Limit
12	Axis 4 - Home Limit	24	Axis 8 - Home Limit

Example:
```
IF(LIM=b11X1)     ; If both limit inputs on axis 1 and the positive-direction
                  ; limit input on axis 2 are active, then do the statements
                  ; between the IF and NIF
TLIM              ; Transfer limit status
NIF               ; End IF statement
```

LIMEN   Limit Input Enable

Type	Inputs; Program Debug Tool	**Product**	**Rev**
Syntax	`<!>LIMEN<d><d><d> .....  (one <d> for each limit input)`	6K	5.0
Units	n/a		
Range	d = 0 (disable, leave off), 1 (disable, leave on),		
	E (enable), or X (don't change)		
Default	E		
Response	`LIMEN:   *LIMENEEE_EEE_EEE_EEE_EEE_EEE_EEE_EEE`		
	`LIMEN.3 *E`		
See Also	HOMLVL, INEN, LH, [LIM], LIMFNC, LIMLVL, TIO, TLIM		

The `LIMEN` command allows you to simulate the activation of specific limit inputs (without actually wiring the inputs to the controller) by disabling them and setting them to a specific level (ON or OFF). This is useful for testing and debugging your program (see program example below). `LIMEN` may only be used for onboard limit inputs (found on the "LIMITS/HOME" connector), not for triggers or external digital inputs. The default state is enabled (`E`), requiring external wiring to exercise their respective `LIMFNC` functions.

Command Syntax:
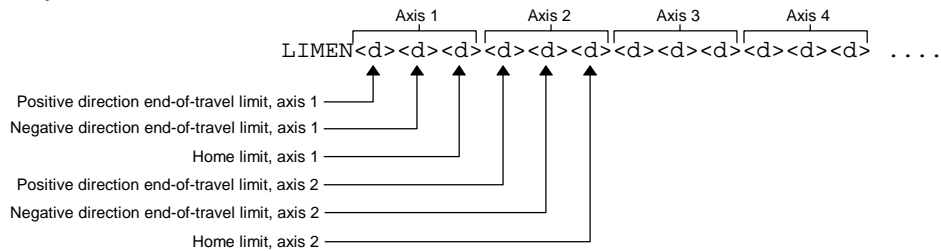
The `LH` command is required to enable checking the state of the end-of-travel limits (i.e., `LIMFNCi-R`, or `LIMFNCi-S`); for example, `LH1` is required to detect the occurrence of the hardware limit activation, as reported with axis status bits 15 and 16 (see `TASF`, `TAS`, `AS`). The default `LH` condition is enabled (`LH1`).

Input bit assignments for the `LIMEN` command vary by product, because of the number of limit inputs available. The input bit patterns for onboard and external I/O bricks are explained on page 6 of this document.

Example:
```
DEL tsting          ; Delete program called "tsting"
DEF tsting          ; Begin definition of program called "tsting"
LIMFNC10-E          ; Define hardware end-of-travel limit #10 (normally defined
                    ; as the positive direction end-of-travel limit for axis 4)
                    ; as a "pause/resume" input.
COMEXR1             ; Activating the pause/resume input will pause command and
                    ; motion execution, de-activating the pause/resume input
                    ; will resume command and motion execution.
MC111              ; Set axes 1-3 to continuous motion profiling mode
A15,15,15          ; Set acceleration on axes 1-3 to 15 units/sec/sec
AD5,5,5            ; Set deceleration on axes 1-3 to 5 units/sec/sec
V4,4,4             ; Set velocity on axes 1-3 to 4 revs/sec
GO111              ; Initiate continuous motion on axes 1-3
END                ; End definition of program called "tsting"
; While this program is running and motion is in progress, you can send
; immediate LIMEN commands to simulate the function of the "pause/resume"
; input as follows:
;   1. Start the program by sending the RUN TSTING command to the controller.
;      Axes 1-3 will start moving, all using the same continuous motion profile.
;   2. Send the !LIMEN.10=1 command to the controller. This disables the
;      "pause/resume" input but simulates its activation. Motion and
;      program execution will pause.
;   3. Send the !LIMEN.10=0 command to the controller. This disables the
;      "pause/resume" input but simulates its de-activation. Motion and
;      program execution will resume.
;   4. Send the !LIMEN.10=E command to the controller. This re-enables the
;      "pause/resume" input for normal operation with an external switch
;      or sensor.
;   5. To stop this experiment, send the !K command to the controller.
;      This "kills" program execution and motion on all three axes.
```

LIMFNC   Input Function for Limit Inputs

Type	Inputs; Limits (end of travel); Homing	**Product**	**Rev**
Syntax	`<!>LIMFNC<i>-<<a>c>`	6K	5.0
Units	i = input # on the "LIMITS/HOME" connector (see page 6);		
	a = axis # (or program # for function P);		
	c = function identifier letter		
Range	i = 1-24 (product dependent); a = 1-8 (product dependent);		
	c = A-T		
Default	A		
Response	LIMFNC: *LIMFNC1-A NO FUNCTION - STATUS OFF		
	(repeated for all onboard limit inputs)		
See Also	COMEXR, COMEXS, [ER], ERROR, INDEB, INFNC, INPLC, INSELP,		
	INSTW, INTHW, JOY, JOYAXH, JOYAXL, JOYVH, JOYVL, K, KDRIVE,		
	LH, [LIM], LIMEN, LIMFNC, LIMLVL, PSET, [SS], TER, TIN,		
	TRGFN, TRGLOT, [TRIG], TSS, TSTAT, TTRIG		

The Limit Input Function (LIMFNC) command defines the function of each individual limit input found on the "LIMITS/HOME" connector(s). The factory default configuration is that each dedicated hardware end-of-travel and home limit is assigned to its respective LIMFNC function. That is, axis 1 positive limit is assigned to LIMFNC1-1R, axis 1 negative limit is assigned to LIMFNC2-1S, axis 1 home limit is assigned to LIMFNC3-1T, etc. A limit of 32 limit inputs may be assigned LIMFNC functions; this excludes functions A ("general-purpose") and R, S, and T (end-of-travel and home limit input functions).

Input debounce. By default, the limit inputs are not debounced. However, when a limit input is assigned a function other than its respective LIMFNC function, it is debounced with the Input Debounce Time (INDEB) command setting for I/O brick zero (default is 4 ms). The INDEB debounce is the period of time that the input must be held in a certain state before the controller recognizes it. This directly affects the rate at which the inputs can change state and be recognized. If a limit is once again returned to its respective LIMFNC function, the debounce is removed.

Input bit assignments vary by product. The number of limits inputs and axes available depends on your product (each axis has two end-of-travel limits and one home limit) — see page 6 for details.

Input scan rate: The limit inputs are scanned once per *system update* (2 milliseconds).

Enabling & disabling inputs. Limit inputs assigned an end-of-travel input function (functions R or S described below) are enabled/disabled with the LH command — the default is enabled. Limit input functions may be overridden with the LIMEN command — the default is enabled (no override).

Multitasking. If the LIMFNC command does not include the task identifier (%) prefix, the function affects the task that executes the LIMFNC command. The functions that may be directed to a task with % are: C, D (without an axis specified), E, F, and P (e.g., 2%LIMFNC3-F assigns limit input 3 as a user fault input for task 2). Multiple tasks may share the same input, but the input may only be assigned one function.

Identifier	Function Description
A	**No special function** (general-purpose input)**.** Status can be used with the LIM assignment/comparison operator.
B	**BCD Program Select.** BCD input assignment to programs, lowest numbered input is least significant bit (LSB). BCD values for inputs are as follows:

	BCD Value
Least Significant Bit Value	1
.	2
.	4
.	8
.	10
.	20
.	40
.	80
Most Significant Bit Value	100

Note: If fewer inputs than shown above are defined to be Program Select Inputs, then the highest input number defined as a Program Select Input is the most significant bit.

An input defined as a BCD Program Select Input will not function until the INSELP command has been enabled.

Identifier	Function Description
C	**Kill.** Kills motion on all axes and halts all command processing (refer to K and KDRIVE command descriptions for further details on the *kill* function). This is an edge detection function and is not intended to inhibit motion. To inhibit motion, use the Pause/Resume function (LIMFNCi-E). When enabled with the ERROR command, bit #6 of the TER and ER commands will report the kill status.
<a>D	**Stop.** Stops motion. Axis number is optional; if no axis number is specified, motion is stopped on all axes. If COMEXS is set to zero (COMEXSØ), program execution will be terminated. If COMEXS is set to 1 (COMEXS1), command processing will continue. With COMEXS set to 2 (COMEXS2), program execution is terminated, but the INSELP value is retained. Motion deceleration during the stop is controlled by the AD & ADA commands. If error bit #8 is enabled (e.g., ERROR.8-1), activating a Stop input will set the error bit and cause a branch to the ERRORP program.
E	**Pause/Continue.** If COMEXR is disabled (COMEXRØ), then only command execution pauses, not motion. With COMEXR enabled (COMEXR1), both command and motion execution are paused. After motion stops, you can release the input or issue a continue (!C) command to resume command processing (and motion of in COMEXR1 mode).
F	**User Fault.** Refer to the ERROR command. If error bit #7 is enabled (e.g., ERROR.7-1), activating a User Fault input will set the error bit and cause a branch to the ERRORP program. **CAUTION**: Activating the user fault input sends an !K command to the controller, "killing" motion on all axes (refer to the K command description for ramifications).
G,H	**Reserved**
I	**Alarm Event** - Will cause the 6K controller to set an Alarm Event in the Communications Server over the Ethernet interface. You must first enable the Alarm checking bit for this input-driven alarm (INTHW.23-1). For details on using alarms, refer to the *6K Series Programmer's Guide*.
aJ	**JOG positive-direction** - Will jog the axis specified in a positive-direction. The JOG command must be enabled for this function to work. **Axis number required**.
aK	**JOG negative-direction.** Will jog the axis specified in a negative-direction. The JOG command must be enabled for this function to work. **Axis number required**.
aL	**JOG Speed Select.** Selects the high or low velocity range while jogging. If the input is active, the high jog velocity range will be selected. Axis number is optional. If no axis number is designated, it defaults to all axes.
M	**Joystick Release.** Signals the controller to end joystick operation and resume program execution with the next statement in your program. When the input is open (high), the joystick mode is disabled (joystick mode can be enabled only if the input is closed, and only with the JOY command). When the input is closed (low), joystick mode can be enabled with the JOY command. The process of using Joystick mode is: 1. Assign the "Joystick Release" input function to a programmable input. 2. At the appropriate place in the program, enable joystick control of motion (with the JOY command). (Joystick mode cannot be enabled unless the "Joystick Release" input is closed.) When the JOY command enables joystick mode for the affect axes, program execution stops on those axes (assuming the Continuous Command Execution Mode is disabled with the COMEXCØ command). 3. Use the joystick to move the axes as required. 4. When you are finished using the joystick, open the "Joystick Release" input to disable the joystick mode. This allows program execution to resume with the next statement after the initial JOY command that started the joystick mode.
N	**Joystick Axis Select.** Allows you to control two pairs of axes with one joystick. Use the JOYAXH and JOYAXL commands to assign analog inputs to control specific axes. Opening the Axis Select input (input is high) selects the JOYAXH configuration. Closing the Axis Select input (input is low) selects the JOYAXL configuration. NOTE: When this input is not connected, the JOYAXH configuration is always in effect.

Continued from previous page

O **Joystick Velocity Select.** Allows you to select the velocity for joystick motion. The `JOYVH` and `JOYVL` commands establish two joystick velocities. Opening the Velocity Select input (input is high) selects the `JOYVH` configuration. Closing the Velocity Select input (input is low) selects the `JOYVL` configuration. The `JOYVL` velocity could be used to quickly move to a location, the `JOYVH` velocity could be used for low-speed accurate positioning. NOTE: When this input is not connected, joystick motion always uses the `JOYVH` velocity setting.

iP **Program Select.** One to one correspondence for input vs. program number. The program number comes from the `TDIR` command. The number specified before the program name is the number to specify within this input definition. For example, in the `LIMFNC1-3P` command, 3 is the program number. An input defined as a Program Select Input will not function until the `INSELP` command has been enabled.

Q **Program Security.** Issuing the `LIMFNCi-Q` command enables the *Program Security* feature and assigns the *Program Access* function to the specified programmable input.

The program security feature denies you access to the `DEF`, `DEL`, `ERASE`, `MEMORY`, `LIMFNC`, and `INFNC` commands until you activate the program access input. Being denied access to these commands effectively restricts altering the user memory allocation. If you try to use these commands when program security is active (program access input is not activated), you will receive the error message `*ACCESS DENIED`. *The `LIMFNCi-Q` command is not saved in battery-backed RAM, so you may want to put it in the start-up program (`STARTP`).*

For example, once you issue the `LIMFNC10-Q` command, the positive end-of-travel limit for axis 4 is assigned the program access function and access to the `DEF`, `DEL`, `ERASE`, `MEMORY`, `LIMFNC`, and `INFNC` commands will be denied until you activate the input.

To regain access to these commands without the use of the program access input, you must issue the `LIMEN` command to disable the program security input, make the required user memory changes, and then issue the `LIMEN` command to re-enable the input. For example, if limit input 3 is assigned as the Program Security input, use `LIMEN.3=1` to disable the input and leave it activated, make the necessary user memory changes, and then use `LIMEN.3=E` to re-enable the input.

aR **End-of-Travel Limit, Positive Direction**. This is the factory default function for each dedicated hardware positive-direction end-of-travel limit input found in the "LIMITS" connector(s). If a trigger input or a digital input on an external I/O brick is assigned this function (e.g. `2INFNC1-1R`), then change the respective limit input's function to something else (e.g., change `LIMFNC1-1R` to `LIMFNC1-A`). When an input is assigned this function, it is not debounced.

aS **End-of-Travel Limit, Negative Direction.** This is the factory default function for each dedicated hardware negative-direction end-of-travel limit input found in the "LIMITS/HOME" connector(s). If a trigger input or a digital input on an external I/O brick is assigned this function (e.g. `2INFNC2-1S`), then change the respective limit input's function to something else (e.g., change `LIMFNC2-1S` to `LIMFNC2-A`). When an input is assigned this function, it is not debounced.

aT **Home Limit.** This is the factory default function for each dedicated hardware home limit input found in the "LIMITS/HOME" connector(s). If a trigger input or a digital input on an external I/O brick is assigned this function (e.g. `2INFNC3-1T`), then change the respective limit input's function to something else (e.g., change `LIMFNC3-1T` to `LIMFNC3-A`). When an input is assigned this function, it is not debounced.

Example:
```
LIMFNC10-3D      ; Redefine the positive EOT input for axis 4 (limit input #10)
                 ; to be a stop input for axis 3
```

LIMLVL

Hardware Limit Input Active Level

Type	Limit (End-of-Travel and Homing)	**Product**	**Rev**
Syntax	`<!>LIMLVL<b><b><b><b><b><b><b> ...` (see drawing below)	6Kn	5.0
Units	n/a		
Range	b = 0 (active low: requires n.c. EOT switch & n.o. home switch),		
	1 (active high: requires n.o. EOT switch & n.c. home switch),		
	or X (don't care)		
Default	0		
Response	LIMLVL: *LIMLVL000_000_000_000_000_000_000_000		
See Also	[AS], LH, LIMEN, [LIM], LIMFNC, HOM, TAS, TLIM		

Use the `LIMLVL` command to define the active state of all dedicated hardware end-of-travel and home limits (found on the "LIMITS/HOME" connectors). The default state is active low.

Command Syntax:

```
                          Axis 1         Axis 2        Axis 3
              LIMLVL<b> <b> <b> <b> <b> <b> <b> <b> <b> .......

Positive direction EOT limit, axis 1
Negative direction EOT limit, axis 1
          Home limit, axis 1
Positive direction EOT limit, axis 2
Negative direction EOT limit, axis 2
          Home limit, axis 2
```
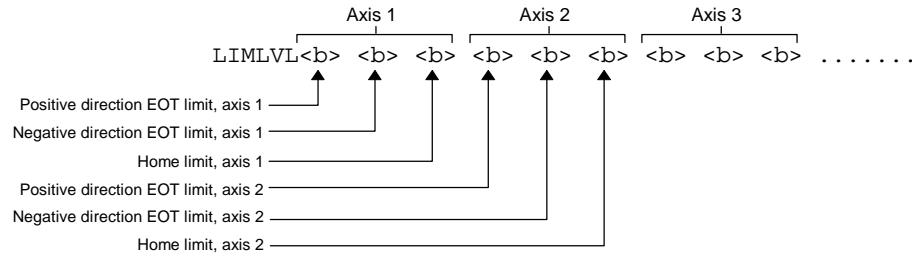
Active Level Setting	Required Switch Type *	State	`LIM`/`TLIM` **Report**
Active low (`LIMLVL0`) *This is the default setting.*	End-of-travel limit: N.C. Home limit: N.O.	Grounded — sinking current (device driving the input is on)	1 (active)
		Not Grounded — not sinking current (device driving the input is off)	0 (inactive)
Active high (`LIMLVL1`)	End-of-travel limit: N.O. Home limit: N.C.	Grounded — sinking current (device driving the input is on)	0 (inactive)
		Not Grounded — not sinking current (device driving the input is off)	1 (active)

* Compumotor recommends that all end-of-travel limit switches be normally-closed, because with normally-closed limit switches the limit function (i.e., inhibit motion) is considered active when the switch contact is open or if the wiring to the switch is broken.

Axis Status (`AS`, `TAS`, and `TASF`) bits 15 and 16 indicate when a hardware end-of-travel limit has been activated (i.e., invoking the "inhibit motion" function).

Wiring instructions and specifications for the limit inputs are provided in your 6K product's *Installation Guide*.

LN   End of Loop

Type	Loops or Program Flow Control	**Product**	**Rev**
Syntax	`<!>LN`	6K	5.0
Units	n/a		
Range	n/a		
Default	n/a		
Response	No response; used in conjunction with the L command		
See Also	L, LX		

The End of Loop (`LN`) command marks the end of a loop. You must use this command in conjunction with the Loop (`L`) command. All buffered commands that you enter between the `L` and `LN` commands are executed as many times as the number that you enter following the `L` command. You may nest loops up to 16 levels deep. **NOTE**: Be careful about performing a `GOTO` between the `L` and `LN` commands. Branching to a different location within the same program will cause the next loop encountered to be nested within the previous loop, unless an `LN` command has already been encountered.

Example:
```
L5         ; Repeat the commands between L and LN five times
GO1110     ; Start motion on axes 1, 2, and 3, axis 4 will remain motionless
LN         ; End loop
```

LOCK   Lock Resource to Task

Type	Multi-tasking	**Product**	**Rev**
Syntax	`<!>LOCK<i,i>`	6K	5.0
Units	1st i = resource number		
	2nd i = 1 (lock the resource) or 0 (unlock the resource)		
Range	1st i = 1 (COM1 port), 2 (COM2 port),		
	or 3 (task swapping)		
	2nd i = 1 (lock the resource) or 0 (unlock the resource)		
Default	0 (= not locked)		
Response	LOCK (see example below)		
See Also	[,], DRPCHK, E, PORT, TSKTRN		

Use the `LOCK` command to make a resource available only to the specified task. The `LOCK`-able resources are:

- COM1 — the "RS-232" communication port or the "ETHERNET" communication port
- COM2 — the "RS-232/485" communication port
- Task Swapping — When task swapping is locked to a specific task, statements in all other tasks will not be executed until the task swapping is again unlocked.

To check the `LOCK` status of all available resources, enter the `LOCK` command without field value. Below is an example response:

```
*LOCK1,0 COM PORT 1    - UNLOCKED
*LOCK2,0 COM PORT 2    - UNLOCKED
*LOCK3,0 TASK SWAPPING - UNLOCKED
```

NOTES

- If one task attempts to lock a resource in a different task (e.g., if Task1 attempts to execute the `2%LOCK1,1` command), the controller will response with an error message ("ALTERNATE TASK NOT ALLOWED").

- If task "A" attempts to lock a resource that is already locked to task "B", command processing in task "A" will pause on the `LOCK` command until task "B" unlocks the resource, at which time task "B" will be able to lock the resource and continue processing.

- A resource may be locked by a task only while that task is executing a program. If program execution is terminated for any reason (e.g., stop, kill, limit, fault, or just reaching the `END` of a program), all resources locked by that task will become unlocked.

Example:
```
LOCK1,1          ; Ensure exclusive COM1 access for the task executing
                 ; this program
WRITE"travel is" ; First part of output string
WRVAR1           ; Numeric value of travel
WRITE"inches."   ; Finish complete string
LOCK1,0          ; Allow other tasks access to COM1
```

LS   Soft Limit Enable

		Product	**Rev**
Type	Limit (End-of-Travel)		
Syntax	`<!><@><a>LS<i>,<i>,<i>,<i>,<i>,<i>,<i>,<i>`	6K	5.0
Units	n/a		
Range	i = 0 (disable both), 1 (disable positive-direction),		
	2 (disable negative-direction) or 3 (enable both)		
Default	0		
Response	LS: *LS0,0,0,0,0,0,0,0		
	1LS: *1LS0		
See Also	[AS], [ER], LSAD, LSADA, LSNEG, LSPOS, TAS, TER, TSTAT		

The Soft Limit Enable (`LS`) command determines the status of the programmable soft move distance limits. With soft limits disabled, motion will not be restricted. After a soft limit absolute position has been programmed (`LSPOS` and `LSNEG`), and the soft limit is enabled (`LS`), a move will be restricted upon reaching the programmed soft limit absolute position. The rate at which motion is decelerated to a stop upon reaching a soft limit is determined by the `LSAD` and `LSADA` commands.

Disable negative- and positive-direction soft limits	i = 0
Enable negative-direction, disable positive-direction soft limit	i = 1
Enable positive-direction, disable negative-direction soft limit	i = 2
Enable negative- and positive-direction soft limits	i = 3

NOTE: The controller maintains an absolute count, even though you may be programming in the incremental mode (`MA0`). The soft limits will also function in incremental mode (`MA0`) or continuous mode (`MC1`). The soft limit position references the commanded position, not the position as measured by the feedback device (e.g., encoder).

NOTE

If a soft limit is encountered while limits are enabled, motion must occur in the opposite direction before a move in the original direction is allowed. You cannot use the `PSET` command to clear the soft limit condition. If limits are disabled, you are free to make a move in either direction.

Example:
```
LSPOS500000,50000   ; Set soft limit positive-direction absolute positions to be
                    ; 500000 units for axis 1, 50000 units for axis 2
                    ; (Soft limits are always absolute)
LSNEG-500000,-50000 ; Set soft limit negative-direction absolute positions to
                    ; be –500000 units for axis 1, -50000 units for axis 2
                    ; (Soft limits are always absolute)
LS3,3               ; Soft limits are enabled on axes 1 and 2
LSAD100,100         ; Soft limit decel set to 100 units/sec/sec on axes 1 and 2
PSET0,0,0,0         ; Set absolute position on all axes to 0
A10,12              ; Set accel to 10 and 12 units/sec/sec for axes 1 and 2
V1,1                ; Set velocity to 1 unit/sec for axes 1 and 2
D100000,1000        ; Set distance to 100000 and 1000 units for axes 1 and 2
GO11XX              ; Initiate motion on axes 1 and 2
```

LSAD   Soft Limit Deceleration

Type	Limit (End-of-Travel)	**Product**	**Rev**
Syntax	`<!><@><a>LSAD<r>,<r>,<r>,<r>,<r>,<r>,<r>,<r>`	6K	5.0
Units	`r = units/sec/sec`		
Range	`0.00001-39,999,998 (depending on the scaling factor)`		
Default	`100.0000`		
Response	`LSAD:    *LSAD100.0000,100.0000,100.0000,100.0000 ...`		
	`1LSAD:   *1LSAD100.0000`		
See Also	`DRES, LHAD, LS, LSADA, LSNEG, LSPOS, SCALE, SCLA`		

The Soft Limit Deceleration (`LSAD`) command determines the value at which to decelerate after a programmed soft limit (`LSPOS` or `LSNEG`) has been hit.

> **UNITS OF MEASURE** and **SCALING**: refer to page 16.

The soft limit deceleration remains set until you change it with a subsequent soft limit deceleration command. Decelerations outside the valid range are flagged as an error, with a message `*INVALID DATA-FIELD x`, where `x` is the field number. When an invalid deceleration is entered the previous deceleration value is retained.

Example: Refer to the soft limit enable (`LS`) command example.

LSADA   Soft Limit Average Deceleration

Type	Motion (S-Curve)	**Product**	**Rev**
Syntax	`<!><@><a>LSADA<r>,<r>,<r>,<r>,<r>,<r>,<r>,<r>`	6K	5.0
Units	`r = units/sec/sec`		
Range	`0.00001-39,999,998 (depending on the scaling factor)`		
Default	`100.0000 (default is a constant deceleration ramp, where LSADA tracks LSAD)`		
Response	`LSADA:   *LSADA100.0000,100.000,100.000,100.000 ...`		
	`1LSADA:  *1LSADA100.0000`		
See Also	`AD, ADA, LS, LSAD, SCALE, SCLA`		

The Soft Limit Average Deceleration (`LSADA`) command allows you to specify the average deceleration for an S-curve deceleration profile when a soft limit is hit. S-curve profiling provides smoother motion control by reducing the rate of change in deceleration; this decel rate of change is known as *jerk*. Refer to page 13 for details on S-curve profiling.

Acceleration scaling (`SCLA`) affects `LSADA` the same as it does for `LSAD`. Refer to page 16 for details on scaling.

Example:
```
LSAD10,10,10,10   ; Sets the maximum deceleration of all four axes
LSADA5,5,7.5,10   ; Sets the average deceleration of all four axes
```

LSNEG   Soft Limit Negative Travel Range

			Product	Rev
Type	Limit (End-of-Travel)		6K	5.0
Syntax	`<!><@><a>LSNEG<r>,<r>,<r>,<r>,<r>,<r>,<r>,<r>`			
Units	`r = units of distance`			
Range	`-999,999,999 - +999,999,999 (scalable)`			
Default	`+0`			
Response	`LSNEG:   *LSNEG+0,+0,+0,+0,+0,+0,+0,+0`			
	`1LSNEG:  *1LSNEG+0`			
See Also	`LS, LSAD, LSADA, LSPOS, PSET, SCALE, SCLD`			

The LSNEG command specifies the distance in absolute units where motion will be restricted when traveling in a negative-travel direction. The reference position used to determine absolute position is set to zero upon power-up, and can be reset using the PSET command. **Be sure to set the LSPOS value greater than the LSNEG value.**

The LSNEG value remains set until you change it with a subsequent LSNEG command.

All soft limit values entered are in absolute steps. If scaling is enabled (SCALE1), LSNEG is internally multiplied by the distance scale factor (SCLD). The soft limit position references the commanded position, not the position as measured by a feedback device (e.g., encoder).

Example: Refer to the soft limit enable (LS) command example.

LSPOS   Soft Limit Positive Travel Range

			Product	Rev
Type	Limit (End-of-Travel)		6K	5.0
Syntax	`<!><@><a>LSPOS<r>,<r>,<r>,<r>,<r>,<r>,<r>,<r>`			
Units	`r = units of distance`			
Range	`-999,999,999 - +999,999,999 (scalable by SCLD)`			
Default	`+0`			
Response	`LSPOS:   *LSPOS+0,+0,+0,+0,+0,+0,+0,+0`			
	`1LSPOS:  *1LSPOS+0`			
See Also	`LS, LSAD, LSADA, LSNEG, PSET, SCALE, SCLD`			

The LSPOS command specifies the distance in absolute units where motion will be restricted when traveling in a positive-travel direction. The reference position used to determine absolute position is set to zero upon power-up, and can be reset using the PSET command. **Be sure to set the LSPOS value greater than the LSNEG value.**

The LSPOS value remains set until you change it with a subsequent LSPOS command.

All soft limit values entered are in absolute steps. If scaling is enabled (SCALE1), LSPOS is internally multiplied by the distance scale factor (SCLD). The soft limit position references the commanded position, not the position as measured by a feedback device (e.g., encoder).

Example: Refer to the soft limit enable (LS) command example.

LX

Terminate Loop

Type	Loops; Program Flow Control		**Product**	**Rev**
Syntax	`<!>LX`		6K	5.0
Units	n/a			
Range	n/a			
Default	n/a			
Response	n/a			
See Also	L, LN, PLN, PLOOP			

The Terminate Loop (`LX`) command terminates the current loop in progress. This command does not halt processing of the commands in the loop until the last command in the current loop iteration is executed. At this point, the loop is terminated. If there are nested loops, only the inner most loop is terminated.

This command can be used externally to terminate the loop only if it is preceded by the immediate command specifier (`!LX`). If the immediate command specifier is not used, the command will have no effect on a loop in progress. An example of where the buffered Terminate Loop command (`LX`) might be used is provided below.

Example:
```
; ****************************************************************
; This program will make the move specified by the GO1110 command
; indefinitely until input 2 goes high, at which point, an LX will
; be issued, terminating the loop.
; ****************************************************************
L0              ; Repeat the commands between L and LN infinitely, or until
                ; a Terminate Loop (LX) command is received
GO1110          ; Start motion on axes 1, 2, and 3,
                ; axis 4 will remain motionless
IF(IN=bX1)      ; If onboard trigger input A2 goes high, execute all
                ; statements between IF and NIF
LX              ; Terminate loop
NIF             ; End IF statement
LN              ; End loop
```

MA   Absolute/Incremental Mode Enable

Type	Motion	**Product**	**Rev**
Syntax	<!><@><a>MA<b><b><b><b><b><b><b><b>	6K	5.0
Units	n/a		
Range	b = 0 (incremental mode) or 1 (absolute mode)		
Default	0		
Response	MA: *MA0000_0000		
	1MA: *1MA0		
See Also	COMEXC, D, GO, GOBUF, PSET		

The Absolute/Incremental Mode Enable (MA) command specifies whether the moves to follow are made with respect to current position (incremental) or with respect to an absolute zero position.

In incremental mode (MAØ), all moves are made with respect to the position at the beginning of the move. This mode is useful for repeating moves of the same distance.

In absolute mode (MA1), all moves are made with respect to the absolute zero position. The absolute zero position is equal to zero upon power up, and can be redefined with the PSET command. An internal counter keeps track of absolute position.

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

Example:
```
PSET0,0,0,1000    ; Set absolute position on axes 1, 2, and 3 to zero,
                  ; and axis 4 to 1000 units
MA1111            ; Enable absolute mode on axes 1 through 4
A2,2,25000,25000  ; Set acceleration to 2, 2, 25000, and 25000 units/sec/sec
                  ; for axes 1, 2, 3 and 4
AD2,2,25000,25000 ; Set deceleration to 2, 2, 25000, and 25000 units/sec/sec
                  ; for axes 1, 2, 3 and 4
V1,1,1,2          ; Set velocity to 1, 1, 1, and 2 units/sec
                  ; for axes 1, 2, 3 and 4 respectively
@D10              ; Set distance on all axes to 10 units
GO1111            ; Initiate motion on all axes (axes 1, 2, and 3 will move
                  ; 10 units in the positive direction, axis 4 will move
                  ; 990 units in the negative direction)
```

MC   **Preset/Continuous Mode Enable**

Type	Motion		**Product**	**Rev**
Syntax	`<!><@><a>MC<b><b><b><b><b><b><b><b>`		6K	5.0
Units	n/a			
Range	b = 0 (preset mode) or 1 (continuous mode)			
Default	0			
Response	MC: *MC0000_0000			
	1MC: *1MC0			
See Also	A, AD, COMEXC, COMEXS, D, FOLMD, [FS], FSHFC, FSHFD, GO, GOBUF, K, MA, PSET, S, SSV, TEST, TFS, V			

The Preset/Continuous Mode Enable (MC) command causes subsequent moves to go a specified distance (MCØ), or a specified velocity (MC1).

In the Preset Mode (MCØ), all moves will go a specific distance. The actual distance traveled is specified by the D, SCLD, and MA commands.

In the Continuous Mode (MC1), all moves will go to a specific velocity with the Distance (D) command establishing the direction (D+ or D-). The actual velocity will be determined by the V and SCLV commands, or the V and DRES commands.

Motion will stop with an immediate Stop (!S) command, an immediate Kill (!K) command, or by specifying a velocity of zero followed by a GO command. Motion can also be stopped with a buffered Stop (S) or Kill (K) command if the continuous command processing mode (COMEXC) is enabled.

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

Example:
```
MA0000                ; Enable incremental mode on all axes
MC0000                ; Enable preset mode on all axes
A2,2,25000,25000      ; Set acceleration to 2, 2, 25000, and 25000 units/sec/sec
                      ; for axes 1, 2, 3 & 4
AD2,2,25000,25000     ; Set deceleration to 2, 2, 25000, and 25000 units/sec/sec
                      ; for axes 1, 2, 3 & 4
V1,1,1,2              ; Set velocity to 1, 1, 1, and 2 units/sec for
                      ; axes 1, 2, 3 & 4 respectively
D10,10,10,10          ; Set distance on all axes to 10 units
GO1111                ; Initiate motion on all axes (axes 1,2, 3, & 4 will
                      ; all move 10 units positive-direction)
COMEXC1               ; Enable continuous command processing mode
MC1111                ; Enable continuous mode on all axes
A8,8,2000,2000        ; Set acceleration to 8, 8, 2000, and 2000 units/sec/sec for
                      ; axes 1, 2, 3 & 4
AD8,8,2000,2000       ; Set deceleration to 8, 8, 2000, and 2000 units/sec/sec for
                      ; axes 1, 2, 3 & 4
V5,5,5,9              ; Set velocity to 5, 5, 5, and 9 units/sec for axes 1, 2, 3 & 4
GO1111                ; Initiate motion on all axes (axes 1,2, and 3 will each
                      ; travel at a velocity of 1 unit/sec, axis 4 will travel
                      ; at a velocity of 2 units/sec)
T15                   ; Wait 15 seconds
@V5                   ; Set velocity to 5 units/sec (axis 4 only affected axis)
GO1111                ; Initiate motion with new velocity of 5 units/sec (all axes)
T8                    ; Wait 8 seconds
@V0                   ; Set velocity to zero
GO1111                ; Initiate motion with new velocity of 5 units/sec (all axes)
WAIT(MOV=b0000)       ; Wait for motion to come to a halt on all axes
COMEXC0               ; Disable continuous command processing mode
```

MEMORY   Partition User Memory

		Product	**Rev**
Type	`Controller Configuration`	6K	5.0
Syntax	`<!>MEMORY<i>,<i>`		
Units	`i = bytes of memory (use even number only)`		
	`1st <i> = partition for "Programs"`		
	`2nd <i> = partition for "Compiled Profiles"`		
Range	`(see table below)`		
Default	`(see table below)`		
Response	`MEMORY:  *MEMORY149000,1000`		
See Also	`[ DATP ], DEF, GOBUF, PCOMP, PLCP, [ SEG ], [ SS ], TDIR,`		
	`TMEM, TSEG, TSS`		

Your controller's memory has two partitions: one for storing *programs* and one for storing *compiled profiles & PLC programs*. The allocation of memory to these two areas is controlled with the MEMORY command.

"Programs" vs. "Compiled Profiles & Programs"

Programs are defined with the DEF and END commands, as demonstrated in the "Program Development Scenario" in the *Programmer's Guide*.

Compiled Profiles & PLC Programs are defined like programs (using the DEF and END commands), but are compiled with the PCOMP command and executed with the PRUN command (but PLCP programs are usually executed with SCANP). Compiled profiles/programs could be a multi-axis *contour* (a series of arcs and lines), an *individual axis profile* (a series of GOBUF commands), a *compound profile* (combination of multi-axis contours and individual axis profiles), or a *PLC program* (for PLC Scan Mode).

Programs intended to be compiled are stored in program memory. After they are compiled with the PCOMP command, they remain in program memory and the *segments* (see diagram below) from the compiled program are stored in compiled memory. The TDIR report indicates which programs are compiled as compiled profiles ("COMPILED AS A PATH") and which programs are compiled as PLC programs ("COMPILED AS A PLC PROGRAM") .

For more information on multi-axis contours (Contouring), compiled profiles for individual axes (Compiled Motion Profiling), and PLC Scan Mode, refer to the *Programmer's Guide*.

MEMORY **Syntax:**

MEMORY80000,70000

Memory allocation for Programs (bytes). Storage requirements depend on the number of ASCII characters in the program.

Memory allocation for Compiled Profiles & Programs (bytes). Storage requirements depend on the number of segmants (1 segment consumes 72 bytes). A segment could be one of these commands:

Contouring:	Compiled Motion:	PLC (PLCP) Program:
PARCM	GOBUF *	IF **
PARCOM	PLOOP	ELSE
PARCOP	GOWHEN	NIF
PARCP	TRGFN	L
PLIN	POUTA	LN
	POUTB	OUT
	POUTC	ANO
	POUTD	EXE
	POUTE	PEXE
	POUTF	VARI **
	POUTG	VARB **
	POUTH	

* GOBUF commands may require up to 4 segments.
** IF statements require at least 2 segments; each AND or OR compound requires an additional segment. VARI and VARB each require 2 segments.

Allocation Defaults and Limits (by Product):

The following table identifies memory allocation defaults and limits for 6K Series products. When specifying the memory allocation, use only even numbers. The minimum storage capacity for one partition area (program or compiled) is 1,000 bytes.

Feature	6K
Total memory (bytes)	150,000
Default allocation (program,compiled)	149000,1000
Maximum allocation for programs	149000,1000
Maximum allocation for compiled profiles/programs	1000,149000
Max. # of programs	400
Max. # of labels	600
Max. # of compiled profiles	300
Max. # of compiled profile segments	2069
Max. # of numeric variables	225
Max. # of integer variables	225
Max. # of string variables	25
Max. # of binary variables	125

When teaching variable data to a data program (DATP), be aware that the memory required for each data statement of four data points (43 bytes) is taken from the memory allocation for program storage.

CAUTION

Issuing a memory allocation command (e.g., `MEMORY80000,70000`) will erase all existing programs and compiled segments. However, issuing the `MEMORY` command by itself (e.g., type `MEMORY` <cr> by itself to request the status of how the memory is allocated) will not affect existing programs or compiled segments.

Checking Memory Status:

To find out what programs reside in your controller's memory, and how much of the available memory is allocated for programs and compiled profile segments, issue the `TDIR` command (see example response below). Entering the `TMEM` command or the `MEMORY` command (<u>without parameters</u>) will also report the available memory for programs and compiled profile segments.

Sample response to `TDIR` command:

```
*1 - SETUP USES 345 BYTES
*2 - PIKPRT USES 333 BYTES
*32322 OF 80000 BYTES (98%) PROGRAM MEMORY REMAINING
*70000 OF 70000 SEGMENTS (100%) COMPILED MEMORY REMAINING
```

Two system status bits (reported with the `TSS`, `TSSF` and `SS` commands) are available to check when compiled profile segment storage is 75% full or 100% full. System status bit #29 is set when segment storage reaches 75% of capacity; bit #30 indicates when segment storage is 100% full.

Example:

```
MEMORY80000,70000 ; Set aside 80,000 bytes for program storage,
                  ; 70,000 bytes for compiled profile segments
```

www.comoso.com

MEPOL   Master Encoder Polarity

Type	Encoder; Following; Controller Configuration		**Product**	**Rev**
Syntax	`<!>MEPOL<b>`		6K	5.0
Units	b = polarity bit			
Range	b = 0 (normal polarity), 1 (reverse polarity), or X (don't care)			
Default	0			
Response	MEPOL: *MEPOL0			
See Also	ENCPOL, [PCME], [PCMS], [PMAS], [PME], PMESET, TPCME, TPME, TPCMS, TPMAS			

Use the `MEPOL` command to reverse the counting direction (polarity) of the Master Encoder input (the encoder connector labeled "Master Encoder"). This allows you to reverse the counting direction without having to change the actual wiring to the encoder input.

Immediately after issuing the `MEPOL` command, the master encoder will start counting in the opposite direction (including all master encoder position registers).

The `MEPOL` command is automatically saved in non-volatile RAM.

MESND   Master Encoder Step and Direction Mode

Type	Encoder; Counter; Following		**Product**	**Rev**
Syntax	`<!>MESND<b>`		6K	5.0
Units	b = enable bit			
Range	b = 0 (quadrature signal), 1 (step & direction), or X (don't care)			
Default	0			
Response	MESND: *MESND0			
See Also	ENCSND, [PME], TPME			

Use the `MESND` command to specify the functionality of the Master Encoder input.

MESND0 .... (default setting) accept a quadrature signal from the master encoder.

MESND1 .... 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-.

[MOV]   Axis Moving 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	[AS], GO, TAS			

The Axis Moving Status (`MOV`) command is used to assign the moving status 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, Ø, 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 Ø through 9.

The axis moving status is also reported with bit #1 of the `TAS`, `TASF` and `AS` commands

Syntax: `VARBn=MOV` where `n` is the binary variable number,
or `MOV` can be used in an expression such as `IF(MOV=b1XX1)`, or `IF(MOV=h3)`

Each bit of the `MOV` command corresponds to a specific axis. The first bit (left to right) is for axis 1, the second is for axis 2, etc. If the specific axis is in motion, the bit will be a one (1). If the specific axis is not in motion, the bit will be a zero (Ø).

Each 6K Series product has 1 moving/not moving bit per axis. For example, the 6K4 has 4 axes, thus 4 moving/not moving bits. If it is desired to assign only one moving/not moving bit to a binary variable, instead of all the moving/not moving bits, the bit select (`.`) operator can be used. The bit select operator, in conjunction with the moving/not moving bit number, are used to specify a specific moving/not moving bit. For example, `VARB1=MOV.2` assigns bit 2 (representing axis 2 moving/not moving) to binary variable 1.

Example:
```
COMEXC1              ; Enable continuous command processing mode
COMEXS1              ; Save command buffer on stop
MC1111               ; Enable continuous mode on all axes
A2,2,25000,25000     ; Set acceleration to 2, 2, 25000, and 25000 units/sec/sec
                     ; for axes 1, 2, 3 and 4 respectively
AD2,2,25000,25000    ; Set deceleration to 2, 2, 25000, and 25000 units/sec/sec
                     ; for axes 1, 2, 3 and 4 respectively
V1,1,1,2             ; Set velocity to 1, 1, 1, and 2 units/sec for axes 1, 2, 3
                     ; and 4 respectively
GO1111               ; Initiate motion on all axes (axes 1,2, and 3 will each
                     ; travel at a velocity of 1 unit/sec, axis 4 will travel
                     ; at a velocity of 2 units/sec)
T5                   ; Wait 5 seconds
S1111                ; Stop motion on all axes
WAIT(MOV=b0000)      ; Wait for motion to come to a halt on all axes
COMEXC0              ; Disable continuous command processing mode
```

NIF   End IF Statement

Type	Program Flow Control or Conditional Branching	**Product**	**Rev**
Syntax	`<!>NIF`	6K	5.0
Units	n/a		
Range	n/a		
Default	n/a		
Response	No response when used in conjunction with the IF command		
See Also	ELSE, IF		

This command is used in conjunction with the `IF` and `ELSE` commands to provide conditional program flow. 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 between the `ELSE` and 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.

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

NOTE: Be careful about performing a `GOTO` between `IF` and `NIF`. Branching to a different location within the same program will cause the next `IF` statement encountered to be nested within the previous `IF` statement, unless an `NIF` command has already been encountered.

Example:
```
IF(IN=b1X0)     ; Specify IF condition to be onboard input 1 = 1, input 3 = Ø
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
```

[NMCY]   Master Cycle Number

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	FMCLEN, FMCNEW, FMCP, [FS], [PMAS], TFS, TNMCY, TRGFN			

The Master Cycle Number (NMCY) command is used to assign the current master cycle number (specific to one axis) to a numeric variable, or to make a comparison against another value. <u>The master must be assigned first (FOLMAS command) before this command will be useful.</u> For a complete discussion of master cycles, refer to the Following chapter in the *Programmer's Guide.*

The value represents the current cycle number, not the position of the master (or the follower). The master cycle number is set to zero when master cycle counting is restarted, and is incremented each time a master cycle finishes (i.e., rollover occurs). It will often correspond to the number of complete parts in a production run. This value may be used for subsequent decision making, or simply recording the cycle number corresponding to some other event.

Syntax: VARn=aNMCY where "n" is the variable number and "a" is the axis number, or NMCY can be used in an expression such as IF(1NMCY>=5). The NMCY command must be used with an axis specifier, or it will default to axis 1 (e.g., VAR1=1NMCY, IF(2NMCY>12), etc.).

Example:
```
IF(2NMCY>500)   ; If the master for axis 2 has moved through 500 cycles ...
WRITE"500 cycles have occurred"  ; Send string to serial port or the AT-bus
NIF             ; End of IF statement
VAR12=3NMCY     ; Set VAR12 to equal the number of cycles that have
                ; occurred on axis 3 master
```

[NOT]   Not

Type	Operator (Logical)		**Product**	**Rev**
Syntax	See below		6K	5.0
Units	n/a			
Range	n/a			
Default	n/a			
Response	n/a			
See Also	[AND], IF, NWHILE, [OR], REPEAT, UNTIL, WAIT, WHILE			

The NOT operator is used in conjunction with the program flow control commands (IF, REPEAT..UNTIL, WHILE..NWHILE, WAIT). The NOT operator compliments a logical expression. If an expression is true, the NOT operator will make the expression false. If an expression is false, the NOT operator will make the expression true. This fact is best illustrated by the following examples:

> If variable #1 equals 1, then the following is a true statement: IF(VAR1<3)
> By using the NOT operator, the same statement becomes false: IF (NOT VAR1<3)
> If variable #2 equals 2, then the following statement is false: WHILE(VAR2=3)
> By using the NOT operator, the same statement becomes true: WHILE (NOT VAR2=3)

To evaluate an expression (NOT Expression) to determine if the expression is true, use the following rule:

> NOT TRUE = FALSE
> NOT FALSE = TRUE

In the following example, variable #1 is displayed, then is incremented by 1 as long as VAR1 is not equal to 10.

Example:
```
VAR1=1              ; Set variable 1 equal to 1
WHILE(NOT VAR1=10)  ; Compare variable 1 to 10, and logically not the expression
WRVAR1              ; Write out variable 1
VAR1=VAR1 + 1       ; Set variable 1 to increment 1 by 1
NWHILE              ; End WHILE statement
```

NTADDR   Ethernet IP Address

Type	Communication Interface	**Product**	**Rev**
Syntax	NTADDR<i,i,i,i>	6K	5.0
Units	i,i,i,i = IP address (commas are used in place of periods)		
Range	i = 0-255		
Default	192,168,10,30 (network address is 192.168.10.30)		
Response	NTADDR: *192,168,10,30		
See Also	TNTMAC		

Use the NTADDR command to change the 6K controller's IP address (e.g., to correct an IP address conflict). **NOTE**: The 6K product needs to be reset (cycle power or issue RESET command) in order for the new address to take effect.

The NTADDR setting is automatically saved in battery backed RAM.

NTMASK   Ethernet Network Mask

Type	Communication Interface	**Product**	**Rev**
Syntax	NTMASK<i,i,i,i>	6K	5.0
Units	i,i,i,i = mask		
Range	i = 0-255		
Default	255,255,255,0		
Response	NTMASK: *255,255,255,0		
See Also	NTADDR, TNTMAC		

Use the NTMASK command to configure the 6K controller's network mask. **NOTE**: The 6K product needs to be reset (cycle power or issue RESET command) in order for the new network mask to take effect.

The NTMASK setting is automatically saved in battery backed RAM.

NWHILE   End WHILE Statement

Type	Program Flow Control or Conditional Branching	**Product**	**Rev**
Syntax	<!>NWHILE	6K	5.0
Units	n/a		
Range	n/a		
Default	n/a		
Response	No response when used in conjunction with the WHILE command		
See Also	WHILE		

The WHILE command, in conjunction with the NWHILE command, provide a means of conditional program flow. The WHILE command marks the beginning of the conditional statement, the NWHILE command marks the end. If the expression contained within the parenthesis of the WHILE command evaluates true, then the commands between the WHILE and NWHILE are executed, and continue to execute as long as the expression evaluates true. If the expression evaluates false, then program execution jumps to the first command after the NWHILE.

Up to 16 levels of WHILE .... NWHILE commands may be nested.

NOTE: Be careful about performing a GOTO between WHILE and NWHILE. Branching to a different location within the same program will cause the next WHILE statement encountered to be nested within the previous WHILE statement, unless an NWHILE command has already been encountered.

Programming order: WHILE(expression) ...commands... NWHILE

Example:
```
WHILE(IN=b1X0)  ; While input 1 = 1, input 3 = Ø, execute commands between
                ; WHILE and NWHILE
T5              ; Wait 5 seconds
TPE            ; Transfer position of all encoders
NWHILE         ; End WHILE statement
```

ONCOND   On Condition Enable

Type	On Condition (Program Interrupt)	**Product**	**Rev**
Syntax	<!><%><@>ONCOND<b><b><b><b>	6K	5.0
Units	n/a		
Range	b = 0 (disable), 1 (enable) or X (don't change)		
Default	0		
Response	ONCOND: *ONCOND0000		
See Also	FSHFD, ONIN, ONP, ONUS, ONVARA, ONVARB, [SS], TSS		

The On Condition Enable (ONCOND) command enables the ONIN, ONUS, ONVARA, and ONVARB commands. When enabled, the expressions specified in the ONIN, ONUS, ONVARA, and ONVARB commands will be continuously evaluated. If any of the expressions ever evaluate true, a GOSUB will be made to the ONP program/subroutine.

ONP, ONIN, ONUS, ONVARA, and ONVARB should be defined before enabling the On Condition. If ONP is not defined first, the error message *UNDEFINED LABEL will appear.

ONCONDbbbb: First b = ONIN Enable
                Second b = ONUS Enable
                Third b = ONVARA Enable
                Fourth b = ONVARB Enable

When ON conditions WILL NOT interrupt immediately: These are situations in which an ON condition does not immediately interrupt the program in progress. However, the fact that the ON condition evaluated true is retained, and when the condition listed below is no longer preventing the interrupt, the interrupt will occur.

- While a WAIT statement is in progress
- While a time delay (T) is in progress
- While a program is being defined (DEF)
- While a pause (PS) is in progress
- While a data read (DREAD, DREADF, or READ) is in progress
- While motion is in progress due to GO, GOL, GOWHEN, HOM, JOY, JOG, or PRUN and the continuous command execution mode is disabled (COMEXCØ).

Multi-Tasking: Each task has it own ONP Program and its own set of On conditions.

Example:
```
DEF bigmov       ; Define program bigmov
D20,20,1,3       ; Sets move distance on axes 1 and 2 to 20 units,
                 ; axis 3 to 1 unit, and axis 4 to 3 units
GO1111           ; Initiate motion on all axes
END              ; End program definition
ONP bigmov       ; Set ON program to bigmov
2ONINxxx1        ; When input #4 on I/O brick 2 is activated,
                 ; GOSUB to the ONP program
ONCOND1000       ; Enable ONIN condition
;
;  Now that the ONP program named bigmov is defined, if input #4 becomes
;  active during normal program operation, the program will GOSUB to the
;  ONP program (bigmov).
```

ONIN   On an Input Condition Gosub

Type	On Condition (Program Interrupt)	**Product**	**Rev**
Syntax	`<!><%><B>ONIN<b><b><b>...<b><b>`	6K	5.0
Units	n/a		
Range	b = 0 (disable), 1 (enable) or X (don't care)		
Default	0		
Response	ONIN: *ONIN0000_0000_0000_0000_0		
	1ONIN: *1ONIN0000_0000_0000_0000_0000_0000_0000_0000		
See Also	INFNC, ONCOND, ONP, TIN		

The On an Input Condition Gosub (`ONIN`) command specifies the input bit pattern which will cause a branch to the ON program (`ONP`). If the input pattern occurs, a GOSUB is performed. The subroutine or program that the GOSUB branches to is selected with the ON program (`ONP`) command.

The number of onboard and external inputs available varies by the product and configuration of I/O bricks used. Refer to page 6 for details.

The `ONIN` command must be enabled using the `ONCOND` command before any branching will occur. Once a branch to the `ONP` program occurs, `ONIN` command will not call the `ONP` program while the `ONP` program is executing, eliminating the possibility of recursive calls. After returning from the `ONP` program, the input pattern specified by the `ONIN` command must evaluate false before another branch to the `ONP` program, resulting from the `ONIN` inputs, will be allowed.

Multi-Tasking: Each task has it own `ONP` Program and its own set of On conditions. Only 1 `ONIN` condition is allowed per task. Therefore, only one I/O brick can be referenced in an `ONIN` condition for a specific task.

Example:
```
DEF bigmov        ; Define program bigmov
D20,20,1,3        ; Sets move distance on axes 1 and 2 to 20 units,
                  ; axis 3 to 1 unit, and axis 4 to 3 units
GO1111            ; Initiate motion on all axes
END               ; End program definition
ONP bigmov        ; Set ON program to bigmov
2ONINxxx11xx1     ; When inputs 4, 5, and 8 on I/O brick 2 is activate,
                  ; GOSUB to the ONP program
ONCOND1000        ; Enable ONIN condition
;
; Now that the ONP program named bigmov is defined, if input #4 becomes
; active during normal program operation, the program will GOSUB to
; the ONP program (bigmov).
```


ONP   On Condition Program Assignment

Type	On Condition (Program Interrupt)	**Product**	**Rev**
Syntax	`<!><%>ONP<t>`	6K	5.0
Units	t = text (name of On Condition program)		
Range	text name of 6 characters or less		
Default	n/a		
Response	ONP: *ONP bigmov		
See Also	DEF, END, ONCOND, ONIN, ONUS, ONVARA, ONVARB		

The On Condition Program (`ONP`) command assigns the program to which programming will GOSUB when an ON condition is met. The program must be defined (`DEF`) previous to the execution of the `ONP` command. The `ONP` command must be specified before enabling the ON conditions (`ONCOND`). If `ONP` is not defined first, the error message `*UNDEFINED LABEL` will appear.

To unassign the program as the ON condition program, issue the `ONP CLR` command. Deleting the program with the `DEL` command will accomplish the same thing.

Within the `ONP` program, the programmer is responsible for checking which ON condition caused the branch, if multiple ON conditions (`ONCOND`) have been enabled. Once a branch to the `ONP` program occurs,

the ONP program will not be called again until after it has finished executing. After returning from the ONP program, the condition that caused the branch must evaluate false before another branch to the ONP program will be allowed.

Multi-Tasking: Each task has it own ONP Program and its own set of On conditions.

Example:
```
DEF bigmov        ; Define program bigmov
D20,20,1,3        ; Sets move distance on axes 1 and 2 to 20 units,
                  ; axis 3 to 1 unit, and axis 4 to 3 units
GO1111            ; Initiate motion on all axes
END               ; End program definition
ONP bigmov        ; Set ON program to bigmov
2ONIN.4-1         ; When input #4 on I/O brick 2 is activated,
                  ; GOSUB to the ONP program
ONCOND1000        ; Enable ONIN condition
;
; Now that the ONP program named bigmov is defined, if input #4 becomes
; active during normal program operation, the program will GOSUB to
; the ONP program (bigmov).
```

ONUS   On a User Status Condition Gosub

Type	On Condition (Program Interrupt)	**Product**	**Rev**
Syntax	<!><%>ONUS<b><b><b>...<b><b> (16 bits)	6K	5.0
Units	n/a		
Range	b = 0 (disable), 1 (enable) or X (don't care)		
Default	0		
Response	ONUS: *ONUS0000_0000_0000_0000		
See Also	INDUSE, INDUST, ONCOND, ONP		

The On a User Status Condition Gosub (ONUS) command specifies the user status bit pattern, defined using the INDUST command, which will cause a branch to the ON program (ONP). If the bit pattern occurs, a GOSUB is performed. The subroutine or program that the GOSUB branches to is selected by the ON program (ONP) command.

The ONUS command must be enabled using the ONCOND command before any branching will occur. Once a branch to the ONP program occurs, ONUS command will not call the ONP program while the ONP program is executing, eliminating the possibility of recursive calls. After returning from the ONP program, the user status bit pattern specified by the ONUS command must evaluate false before another branch to the ONP program, resulting from the ONUS status bits, will be allowed.

Multi-Tasking: Each task has it own ONP Program and its own set of On conditions.

Example:
```
INDUSE1           ; Enable user status
INDUST1-5A        ; User status bit 1 defined as axis 1 status bit 5
INDUST2-3F        ; User status bit 2 defined as axis 6 status bit 3
3INDUST3-5J       ; User status bit 3 defined as input 5 on I/O brick 3
INDUST4-1K        ; User status bit 4 defined as interrupt status bit 1
2%INDUST16-2I     ; User status bit 16 defined as system status bit 2 for task 2
DEF bigmov        ; Define program bigmov
D20,20,1,3        ; Sets move distance on axes 1 and 2 to 20 units,
                  ; axis 3 to 1 unit, and axis 4 to 3 units
GO1111            ; Initiate motion on axes 1-4
END               ; End program definition
ONP bigmov        ; Set ON program to bigmov
ONUSxxx1          ; On user status bit #4 (interrupt status bit 1) GOSUB to
                  ; the ONP program
ONCOND0100        ; Enable ONUS condition
```

ONVARA   On Variable 1 Condition Gosub

Type	On Condition (Program Interrupt)	**Product**	**Rev**
Syntax	`<!><%>ONVARA<i,i,i>`	6K	5.0
Units	See below		
Range	±999,999,999.99999999		
Default	+0.0,+0.0,+0.0		
Response	ONVARA: *ONVARA+0.0,+0.0,+0.0		
See Also	ONCOND, ONP, ONVARB, VAR, VARI		

The On Variable 1 Condition Gosub (ONVARA) command specifies the low and high values which will cause a branch to the ON program (ONP). If the value of variable 1 is less than or equal to the first `i`, or greater than or equal to the second `i`, a GOSUB is performed. The subroutine or program that the GOSUB branches to is selected by the ON program (ONP) command. If the third field is non-zero, integer variables (VARI) are used for the comparison.

The ONVARA command must be enabled using the ONCOND command before any branching will occur. Once a branch to the ONP program occurs, ONVARA command will not call the ONP program while the ONP program is executing, eliminating the possibility of recursive calls. After returning from the ONP program, variable 1 must be reset to a value within the low and high values before another branch to the ONP program, resulting from the value of variable 1, will be allowed.

Multi-Tasking: Each task has it own ONP Program and its own set of On conditions.

Example:
```
DEF bigmov          ; Define program bigmov
D20,20,1,3          ; Sets move distance on axes 1 and 2 to 20 units,
                    ; axis 3 to 1 unit, and axis 4 to 3 units
GO1111              ; Initiate motion on all axes
END                 ; End program definition
ONP bigmov          ; Set ON program to bigmov
ONVARA0,12          ; On VAR1 <= 0, or VAR1 >= 12 GOSUB to ONP program
ONCOND0010          ; Enable ONVARA condition
```

ONVARB   On Variable 2 Condition Gosub

Type	On Condition (Program Interrupt)	**Product**	**Rev**
Syntax	`<!><%>ONVARB<i,i,i>`	6K	5.0
Units	See below		
Range	±999,999,999.99999999		
Default	+0.0,+0.0,+0.0		
Response	ONVARB: *ONVARB+0.0,+0.0,+0.0		
See Also	ONCOND, ONP, ONVARA, VAR, VARI		

The ONVARB command specifies the low and high values which will cause a branch to the ON program (ONP). If the value of variable 2 is less than or equal to the first `i`, or greater than or equal to the second `i`, a GOSUB is performed. The subroutine or program that the GOSUB branches to is selected by the ON program (ONP) command. If the third field is non-zero, integer variables (VARI) are used for the comparison.

The ONVARB command must be enabled using the ONCOND command before any branching will occur. Once a branch to the ONP program occurs, ONVARB command will not call the ONP program while the ONP program is executing, eliminating the possibility of recursive calls. After returning from the ONP program, variable 2 must be reset to a value within the low and high values before another branch to the ONP program, resulting from the value of variable 1, will be allowed.

Multi-Tasking: Each task has it own ONP Program and its own set of On conditions.

Example:
```
DEF bigmov          ; Define program bigmov
D20,20,1,3          ; Sets move distance on axes 1 and 2 to 20 units,
                    ; axis 3 to 1 unit, and axis 4 to 3 units
GO1111              ; Initiate motion on all axes
END                 ; End program definition
ONP bigmov          ; Set ON program to bigmov
ONVARB0,12          ; On VAR2 <= 0, or VAR2 >= 12 GOSUB to ONP program
ONCOND0001          ; Enable ONVARB condition
```

[OR]   Or

Type	Operator (Logical)	**Product**	**Rev**
Syntax	See below	6K	5.0
Units	n/a		
Range	n/a		
Default	n/a		
Response	n/a		
See Also	[AND], IF, [NOT], NWHILE, REPEAT, UNTIL, WAIT, WHILE		

Use the OR command as a logical operator in a program flow control command (IF, REPEAT, UNTIL, WHILE, NWHILE, WAIT). The OR command logically links two expressions. If either of the two expressions are true, and are linked with an OR command, then the whole statement is true. This fact is best illustrated by example.

If VAR1=1 and VAR2=1 then, even though variable 2 is not greater than 3, this is a true statement: IF(VAR1>0 OR VAR2>3). This statement would <u>not</u> be true: IF(VAR1<>1 OR VAR2=2).

To evaluate an expression (Expression 1 OR Expression 2 = Result) to determine if the whole expression is true, use the following rule:

TRUE OR TRUE = TRUE	FALSE OR TRUE = TRUE
TRUE OR FALSE = TRUE	FALSE OR FALSE = FALSE

Example:
```
VAR1=1                  ; Set variable 1 equal to 1
IF(VAR1=1 OR IN=b1XXX)  ; Compare variable 1 to 1, and check for input #1
                        ; to be active
WRITE"FIRST EXAMPLE"    ; If either condition is true, write out FIRST EXAMPLE
NIF                     ; End IF statement
```


OUT   Output State

Type	Output	**Product**	**Rev**
Syntax	<!><B>OUT<b><b><b>...<b><b><b>	6K	5.0
Units	n/a		
Range	b = 0 (off), 1 (on) or X (don't change)		
Default	0		
Response	n/a		
See Also	OUTALL, OUTEN, OUTFNC, OUTLVL, OUTP, TIO, TOUT		

The Output State (OUT) command turns the output bits on and off. You may use this command to control any of the onboard outputs, as well as any outputs on external I/O bricks, <u>as long as they are left in the default function</u> (OUTFNCi-A). If you attempt to change the state of an output that is <u>not</u> defined as an OUTFNCi-A (general-purpose) output, the controller will respond with an error message ("OUTPUT BIT USED AS OUTFNC") and the OUT command will not be executed (but command processing will continue).

The number of onboard and external outputs varies by the product and configuration of I/O bricks used. Refer to page 6 for details.

If it is desired to set only one output value, instead of all outputs, the bit select (.) operator can be used, followed by the number of the specific output. For example, OUT.12-1 turns on output 12.

Example:
```
2OUT10          ; Turn on outputs 1 & 2 on I/O brick 2
1OUT.9-1        ; Turn on output 9 (the 1st I/O point on SIM2) on I/O brick 1
```

[OUT]   Output Status

Type	`Assignment or Comparison`	**Product**	**Rev**
Syntax	`See below`	6K	5.0
Units	`n/a`		
Range	`b = 0 (off), 1 (on) or X (don't change)`		
Default	`0`		
Response	`n/a`		
See Also	`OUTALL, OUTEN, OUTFNC, OUTLVL, TIO, TOUT, VARB`		

Use the Output Status (`OUT`) operator to assign the output states to a binary variable (`VARB`), 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, Ø, 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 Ø through 9.

Syntax: `VARBn=<B>OUT` where "n" is the binary variable number and "<B>" is number of the I/O brick where the output resides (not required if addressing the onboard outputs),
or `OUT` can be used in an expression such as `IF(2OUT=b11Ø1)`, or `IF(1OUT=h7F)`

The number of onboard and external outputs varies by product and number I/O bricks used. Refer to page 6 for details.

The function of the outputs is established with the `OUTFNC` command (although the `OUT` operator looks at all outputs regardless of their assigned function from the `OUTFNC` command). If it is desired to assign only one output value to a binary variable, instead of all outputs, the bit select (`.`) operator can be used, followed by the number of the specific output. For example, `VARB1=2OUT.12` assigns output 12 (the 2nd I/O point on SIM2) on I/O brick 2 to binary variable 1.

Example:
```
VARB1=OUT          ; Output status assigned to binary variable 1
VARB2=OUT.4        ; On-board output bit 4 assigned to binary variable 2
VARB2              ; Response if bit 4 is set to 1 (for 6K4, 6K6, & 6K8):
                   ; *VARB2=XXX1_XXXX
IF(OUT=b110X1)     ; If the output status contains 1's for outputs 1, 2, & 5,
                   ; and a 0 for output 4, do the IF statement
TREV               ; Transfer revision level
NIF                ; End IF statement
```

OUTALL   Output State for Multiple Outputs

Type	Output	**Product**	**Rev**
Syntax	<!><B>OUTALL<i>,<i>,<b>	6K	5.0
Units	1st i = beginning number of output range		
	2nd i = ending number of output range		
	b = enable/disable bit		
Range	1st i = 1 to *n* (*n* is max. number of outputs available)		
	2nd i = First i to *n*		
	b = 0 (off) or 1 (on)		
Default	0		
Response	n/a		
See Also	OUT, OUTEN, OUTFNC, OUTLVL, TIO, TOUT		

The OUTALL command turns a range of output bits on and off. You may use this command to control any contiguous range of the onboard outputs, as well as any outputs on external I/O bricks, <u>as long as all outputs in the range are left in the default function (OUTFNCi-A)</u>. If you attempt to change the state of an output that is <u>not</u> defined as an OUTFNCi-A (general-purpose) output, the controller will respond with an error message ("OUTPUT BIT USED AS OUTFNC") and the OUTALL command will not be executed (but command processing will continue).

The number of onboard and external outputs varies by the product and configuration of I/O bricks used. Refer to page 6 for details.

Example:
```
OUTALL1,4,1     ; Turn on on-board outputs 1-4
2OUTALL3,8,1    ; On I/O brick 2, turn on outputs at I/O locations 3-8
                ; (I/O pins 3-8 on SIM1)
```

OUTEN   Output Enable

Type	Output or Program Debug Tool	**Product**	**Rev**
Syntax	<!><B>OUTEN<d><d><d><d>... (one <d> for each input)	6K	5.0
Units	n/a		
Range	d = 0 (Disable output function and turn output off)		
	d = 1 (Disable output function and turn output on)		
	d = E (Enable output function)		
	d = X (don't change)		
Default	E		
Response	OUTEN: *OUTENEEEE_EEEE (onboard outputs)		
	1OUTEN: *1OUTENEEEE_EEEE_EEEE_EEEE_EEEE_EEEE_EEEE_EEEE		
	1OUTEN.3 *E		
See Also	OUT, OUTFNC, OUTLVL, TIO, TOUT, TSTAT		

The Output Enable (OUTEN) command allows the user to disable any of the outputs from their configured function and set them on or off. This command is used for troubleshooting and initial start-up testing. It allows you to simulate output operations by bypassing the configured output function.

The OUTEN command has no effect on onboard outputs (located on the "TRIGGERS/OUTPUTS" connector) when they are configured as output-on-position outputs with the OUTFNCi-H command.

The number of onboard and external outputs varies by the product and configuration of I/O bricks used. Refer to page 6 for details.

Example:
```
; This allows the user to test if the fault output is working,
; without the inconvenience of trying to force a fault.
1OUTFNC1-1B     ; Define output #1 on I/O brick 1 as axis 1 moving/not moving
1OUTFNC2-2B     ; Define output #2 on I/O brick 1 as axis 2 moving/not moving
1OUTFNC3-A      ; Define output #3 on I/O brick 1 as programmable
1OUTFNC4-A      ; Define output #4 on I/O brick 1 as programmable
1OUTFNC5-F      ; Define output #5 on I/O brick 1 as fault output
1OUTENxxxx1     ; Disable programmed function of output #5 on I/O brick 1
                ; and turns it on
```

OUTFNC   Output Function

Type	`Output`	**Product**	**Rev**
Syntax	`<!><B>OUTFNC<i><-<a>c>`	6K	5.0
Units	`i = output #, a = axis, c = function identifier (letter)`		
Range	`i = 1-32 (I/O brick dependent — see page 6)`		
	`a = 1-8 (depends on product)`		
	`c = A-H`		
Default	`c = A (programmable output function – default)`		
Response	`OUTFNC:    (function and status of onboard outputs)`		
	`1OUTFNC:   (function and status of outputs on I/O brick 1)`		
	`1OUTFNC1: *1OUTFNC1-A PROGRAMMABLE OUTPUT - STATUS OFF`		
See Also	`DRFEN, OUT, OUTEN, OUTLVL, OUTP, OUTPLC, OUTTW, POUT, SMPER,`		
	`TIO, TSTAT`		

The Output Function (`OUTFNC`) command defines the functions for each output. The factory setting for all the outputs is programmable output bits (`OUTFNCi-A`). A limit of 32 output may be assigned `OUTFNC` functions; this excludes `A` ("general-purpose") function.

For the functions that are axis specific (B, D, and E), an optional axis specifier may be placed in front of the function. By placing the axis specifier in front of the function letter, the output will only go active when the specific axis specified has the corresponding condition. If an axis specifier is not specified, then if any of the axes have the corresponding condition, the output will go active. The output functions are as follows:

Output bit assignments vary by product. The number of onboard and external outputs varies by the product and configuration of I/O bricks used. Refer to page 6 for details.

Output Scan Rate: The programmable outputs are scanned once per *system update* (2 milliseconds).

Multitasking. If the `OUTFNC` command does not include the task identifier (`%`) prefix, the function affects the task that executes the `OUTFNC` command. Only function "C" may be directed to a specific task with the `%` prefix (e.g., `2%OUTFNC3-C` assigns onboard output 3 as a program-in-progress output for task 2). Multiple tasks may share the same output, but the output may only be assigned one function.

Identifier	Function Description
A	**Programmable Output**: Standard output (default function). Turn on or off with the `OUT`, `POUTn`, or `OUTALL` commands to affect external processes. To view the state of the outputs, use the `TOUT` command. To use the state of the outputs as a basis for conditional branching or looping statements (`IF`, `REPEAT`, `WHILE`, etc.), use the `[ OUT ]` command.
<a>B	**Moving/Not Moving Axis**: Output activates when the axis is moving. As soon as the move is completed, the output will change to the opposite state. Servo Axes: With the target zone mode enabled (`STRGTE1`), the output will not change state until the move completion criteria set with the `STRGTD` and `STRGTV` commands has been met. In this manner, the output functions as an *In Position* output.
C	**Program in Progress**: Output activates when a program is being executed. After the program is finished, the output's state is reversed.
<a>D	**End-of-Travel Limit Encountered**: Output activates when a hard or soft end-of-travel limit has been encountered. When a limit is encountered, you will not be able to move the motor in that same direction until you clear the limit by changing direction (`D`) and issuing a `GO` command. (An alternative is to disable the limits with the `LH0` command, but this is recommended only if the motor is not coupled to the load.)
<a>E	**Stall Indicator** (Stepper axes only): Output activates when a stall is detected. To detect a stall, you must first connect an encoder and enable stall detection with the `ESTALL1` command. For details refer to the *Programmer's Guide*.
F	**Fault Indicator**: Output activates when either the user fault input or the drive fault input becomes active. The user fault input is a general-purpose input defined as a user fault input with the `INFNCi-F` or `LIMFNCi-F` command. Make sure the drive fault input is enabled (`DRFEN`) and the drive fault active level (`DRFLVL`) is appropriate for the drive you are using.

<a>G   **Position Error Exceeds Max. Limit** (Servos Only): Output activates when the maximum allowable position error, as defined with the `SMPER` command, is exceeded. The position error (`TPER`) is defined as the difference between the commanded position (`TPC`) and the actual position as measured by the feedback device. When the maximum position error is exceeded (usually due to instability or loss of position feedback from the feedback device), the controller shuts down the drive and sets error status bit #12 (reported by the `TER` command). If the `SMPER` command is set to zero (`SMPER0`), the position error will not be monitored; thus, the *Maximum Position Error Exceeded* function will not be usable.

<a>H   **Output On Position**: Output activates when the specified axis is at a specified position (servo axes can use encoder position only; stepper axes can use commanded position or encoder position, depending on the `ENCCNT` setting for that axis). Applicable only to the onboard outputs found on the "**TRIGGERS/OUTPUTS**" connectors. Output On Position function parameters are configured with the `OUTPn` commands.

Example:
```
1OUTFNC1-3B    ; Define output #1 on I/O brick 1 as axis 3 moving/not moving
1OUTFNC2-D     ; Define output #2 on I/O brick 1 to go active when any of
               ; the limits are hit on any axis
```

OUTLVL   Output Active Level

			Product	Rev
Type	`Output`		**6K**	5.0
Syntax	`<!><B>OUTLVL<b><b><b>...<b><b><b>`			
Units	`n/a`			
Range	`b = 0 (active low), 1 (active high) or X (don't change)`			
Default	`0`			
Response	`OUTLVL:    *OUTLVL0000_0000 (onboard outputs)`			
	`1OUTLVL:   *1OUTLVL0000_0000_0000_0000_0000_0000_0000_0000`			
	`1OUTLVL.3  *0`			
See Also	`OUT, OUTEN, OUTFNC, OUTP, OUTPLC, OUTTW, POUT, TOUT`			

The Output Active Level (`OUTLVL`) command defines the active state of each programmable output. The default state is active low. Refer to the 6K Series product *Installation Guide* for programmable output schematics. The `OUTLVL` setting is <u>NOT</u> saved in battery-backed RAM; therefore, on power up or reset, the `OUTLVL` setting will default to the factory default setting (thus, the `OUTLVL` command is a good candidate for inclusion in your `STARTP` program).

The number of onboard and external outputs varies by the product and configuration of I/O bricks used. Refer to page 6 for details.

Using Outputs on Expansion I/O Bricks:
- <u>Sinking vs. Sourcing Outputs</u>. On power up, the 6K controller auto-detects the state of the jumper for each output SIM on each external I/O brick, and automatically changes the `OUTLVL` setting accordingly. If sinking outputs are detected (factory default setting), `OUTLVL` is set to active low; if sourcing outputs are detected, `OUTLVL` is set to active high. For details on the jumper, refer to you product's *Installation Guide*.
- <u>Disconnect I/O Brick</u>. If the I/O brick is disconnected (or if it loses power), the controller will perform a kill (all tasks) and set error bit #18. The controller will remember the brick configuration (volatile memory) in effect at the time the disconnection occurred. When you reconnect the I/O brick, the controller checks to see if anything changed (SIM by SIM) from the state when it was disconnected. If an existing SIM slot is changed (different SIM, vacant SIM slot, or jumper setting), the controller will set the SIM to factory default `INEN` and `OUTLVL` settings. If a new SIM is installed where there was none before, the new SIM is auto-configured to factory defaults.

When an output is defined to be active low, an `OUT1` command will cause a output to be pulled to ground. When an output is defined to be active high, an `OUT1` command will cause a output to source current from the power supply.

Example:
```
OUTLVL1x0  ; Configure onboard output 1 to be active high, output 2 unchanged,
           ; and output 3 as active low
```

OUTP   Output on Position — Axis Specific

Type	`Output`	**Product**	**Rev**
Syntax	`<!>OUTPn<b>,<b>,<r>,<i>`	6K	5.0

Units	`n = axis/output identifier letter`
	`1ˢᵗ & 2ⁿᵈ b = enable/modal bits;`
	`r = scalable distance`
	`i = time (ms)`		

Range	`n = A-H (A for output 1, axis 1, B for output 2, axis 2, etc.)`
	`1ˢᵗ b = 1 (enable output on position) or 0 (disable)`
	`2ⁿᵈ b = 1 (incremental position) or 0 (absolute position)`
	`r = -999,999,999 to +999,999,999`
	`i = 0-65535`		

Default	`0,0,0,0`		
Response	`OUTPA:  *OUTPA0,0,+0,0`		
See Also	`AXSDEF, ENCCNT, [ OUT ], OUT, OUTFNC, PSET, SFB`		

Use the Output on Position (`OUTPn`) command to configure the respective onboard output (located on the "TRIGGERS/OUTPUTS" connectors) to activate based on the specified position of the respective axis. Onboard output 1 corresponds to axis 1, output 2 to axis 2, and so on. The position referenced is dependent upon whether the axis is configured for servo or stepper (see `AXSDEF` command):

- Servo Axes: The referenced position is the encoder position (analog input position cannot be used). Therefore, to use this feature, encoder feedback must be selected with the `SFB` command before the `OUTPn` command is executed. If the `SFB` command is changed, the output-on-position function is disabled until a new `OUTPn` command re-enables the function.

- Stepper Axes: The referenced position depends on the `ENCCNT` setting at the time the `OUTPn` command is executed. If `ENCCNT0` (factory default), the commanded position is used, if `ENCCNT1`, the encoder position is used.

To use the `OUTPn` command, you must first use the `OUTFNCi-H` command to configure the onboard output to function as an *output on position* output. (The "`i`" in the `OUTFNCi-H` command represents the number of the onboard output in the product's output bit pattern — see page 6 for output bit patterns for each product.) Refer to the programming example below.

Syntax:

OUTP n   <b>, <b>, <r>, <i>

Axis/Output Specifier:

- A ... Turn on output 1 based on axis #1's position
- B ... Turn on output 2 based on axis #2's position
- C ... Turn on output 3 based on axis #3's position
- D ... Turn on output 4 based on axis #4's position
- E ... Turn on output 5 based on axis #5's position
- F ... Turn on output 6 based on axis #6's position
- G ... Turn on output 7 based on axis #7's position
- H ... Turn on output 8 based on axis #8's position

Enable Bit:

- 1 ... Enable the output-on-position function
- 0 ... Disable the output-on-position function

Servo Axes: If an `SFB` command is executed, the function is disabled.

Increment or Absolute Position Comparison:

- 1 ... Set position comparison to incremental (measured from the last start-motion command, such as `GO`, `GOL`, `GOWHEN`, etc.)
- 0 ... Set position comparison to absolute

Time (milliseconds):

Time (milliseconds) the output is to stay active. The output activates when the specified position (`<r>`) is reached or exceeded, and stays active for the specified time.

If this field is set to zero, the output will stay active for as long as the actual distance equals or exceeds the position comparison distance (this is possible only for an absolute position comparison).

Position:

Scalable distance (distance is either incremental or absolute, depending on the second `<b>` data field).

Servo Axes:
 Only the encoder position can be used.

Stepper Axes:
 · If `ENCCNT0`, the commanded position is used.
 · If `ENCCNT1`, the encoder position is used.

NOTE

The output activates only during motion; therefore, issuing a `PSET` command to set the absolute position counter to activate the output on position will not turn on the output until the next motion occurs.

Example (servo axes)**:**

```
AXSDEF10            ; Define axis 1 as servo, axis 2 as stepper
SFB1                ; Select encoder feedback for axis 1
OUTFNC1-H           ; Set onboard output #1 as an "output on position" output
OUTFNC2-H           ; Set onboard output #2 as an "output on position" output
OUTPA1,0,+50000,50  ; Turn on onboard output #1 for 50 ms when the encoder
                    ; position of axis #1 is > or = absolute position +50,000
OUTPB1,1,+30000,50  ; Turn on onboard output #2 for 50 ms when the axis #2's
                    ; commanded position reaches > or = incremental position
                    ; 30,000 (since the last GO)
```

OUTPLC   Establish PLC Strobe Outputs

		Product	Rev
Type	`Output`	6K	5.0
Syntax	`<!><B>OUTPLC<i>,<i-i>,<i>,<i>`		
Units	See below		
Range	See below		
Default	1,0-0,0,0		
Response	`OUTPLC1:    *0-0,0,0 (onboard outputs referenced)` `1OUTPLC1:    *0-0,0,0 (outputs on I/O brick 1 referenced)`		
See Also	`INPLC, OUT, OUTEN, OUTFNC, OUTLVL, OUTTW, [ TW ]`		

The Establish PLC Strobe Outputs (`OUTPLC`) command with its corresponding `INPLC` command configure the applicable inputs and outputs to read data from a parallel I/O device such as a PLC (Programmable Logic Controller), or a passive thumbwheel module. The actual data transfer occurs with the `TW` command. Refer to the `TW` command for a description of the data transfer process.

The `OUTPLC` command has four fields (`<i>`,`<i-i>`,`<i>`,`<i>`):

Data Field	Description
Field 1: `<i>`	**Set #**: There are 4 possible `OUTPLC` sets (1-4). This field identifies which set to use.
Field 2: `<i-i>`	**Strobe Output #s**: Data reads with the `TW` command are strobed by the outputs selected in this field. The first number is the first output, and the second number is the last output. The outputs must be consecutive. The number of outputs should equal half the number of the maximum number of BCD digits required. If 6 digits are being read, then three outputs are needed as each output strobe selects two BCD digits. Refer to page 6 for help in identifying which output bits are available to place in this field.
Field 3: `<i>`	`TW` **Command Pending**: This field identifies an output that becomes active on a TW command and then turns off on completion of the `TW` command. This output can signal a device that a `TW` command is pending. A zero in this field will not activate any output.
Field 4: `<i>`	**Strobe Time**: This field identifies the length of time an output will stay active in order to read the BCD digits. The strobe time (in milliseconds) should be greater than the PLC scan time, if a PLC is being used, or set greater than the minimal debounce time if using thumbwheels. Range = 1 - 5000 milliseconds.

To disable a specific PLC set, enter `OUTPLCn,Ø-Ø,Ø,Ø` where `n` is the PLC set (1-4).

Example:
```
INPLC2,1-8,9,10   ; Set INPLC set 2 as BCD digits on onboard inputs 1-8,
                  ; with input 9 as the sign bit, and input 10 as the data valid
OUTPLC2,1-4,5,50  ; Set OUTPLC set 2 as output strobes on onboard outputs 1-4,
                  ; with output 5 as the command pending bit, and strobe time
                  ; of 50 milliseconds
A(TW6)            ; Read data into axis 1 acceleration using INPLC set 2
                  ; and OUTPLC set 2 as the data configuration
```

OUTTW   Establish Thumbwheel Strobe Outputs

Type	`Output`	**Product**	**Rev**
Syntax	`<!><B>OUTTW<i>,<i-i>,<i>,<i>`	6K	5.0
Units	See below		
Range	See below		
Default	`1,0-0,0,0`		
Response	`OUTTW1:   *0-0,0,0 (onboard outputs referenced)`		
	`1OUTTW1:  *0-0,0,0 (outputs on I/O brick 1 referenced)`		
See Also	`INSTW, OUT, OUTEN, OUTFNC, OUTLVL, OUTPLC, [ TW ]`		

The Establish Thumbwheel Strobe Outputs (`OUTTW`) command with its corresponding `INSTW` command configure the applicable inputs and outputs to read data from an active thumbwheel device. The actual data transfer occurs with the `TW` command. Refer to the `TW` command for a description of the data transfer process.

The `OUTTW` command has four fields (`<i>,<i-i>,<i>,<i>`):

Data Field	Description
Field 1: `<i>`	**Set #**: There are 4 possible `OUTTW` sets (1-4). This field identifies which set to use.
Field 2: `<i-i>`	**Strobe Output #s**: Data reads with the `TW` command are strobed by the outputs selected in this field. The first number is the first output, and the second number is the last output. The outputs must be consecutive. The number of outputs should be compatible to the thumbwheel device. Refer to page 6 for help in identifying which output bits are available to place in this field.
Field 3: `<i>`	**Thumbwheel Enable Output**: This field identifies an output that becomes active on a `TW` command and then turns off on completion of the `TW` command. This output can enable a thumbwheel module to respond, thus allowing multiple thumbwheels to be wired to the inputs and outputs. A zero in this field will not activate any output.
Field 4: `<i>`	**Strobe Time**: This field identifies the length of time an output will stay active to read the BCD digits. The strobe time (in milliseconds) should be set to a minimal debounce time. Range = 1-5000 milliseconds.

Example:
```
INSTW2,1-4,5      ; Set INSTW set 2 as BCD digits on onboard inputs 1-4, with
                  ; input 5 as the sign bit
OUTTW2,1-3,4,50   ; Set OUTTW set 2 as output strobes on onboard outputs 1-3,
                  ; with onboard output 4 as the output enable bit, and
                  ; strobe time of 50 milliseconds
A(TW2)            ; Read data into axis 1 acceleration using INSTW set 2 and
                  ; OUTTW set 2 as the data configuration
```

PA   Path Acceleration

Type	Path Contouring or Motion (Linear Interpolated)	**Product**	**Rev**
Syntax	`<!>PA<r>`	6K	5.0
Units	r = units/sec/sec (scalable by SCLD)		
Range	0.00001-39,999,998 (depending on the scaling factor)		
Default	10.0000		
Response	PA: *PA10.0000		
See Also	GOL, PAA, PAD, PADA, SCLD, SCALE		

The Path Acceleration (PA) command specifies the path acceleration to be used with linearly interpolated moves (GOL), and all contouring moves (PLIN, PARCM, PARCOM, PARCOP, PARCP). For both the linear interpolated and the contouring moves, the path acceleration refers to the acceleration experienced by the load as motion gains speed along the path. For linearly interpolated moves, the acceleration of each individual axis is dependent on the distance it contributes to the total path traveled by the load. In contouring paths, the acceleration of each individual axis is dependent on the direction of travel in the X-Y plane. **NOTE:** *The PA value can be altered between path segments, but not within a path segment.*

Contouring and linear interpolation are discussed in detail in the Custom Profiling chapter of the *Programmer's Guide*.

UNITS OF MEASURE and **SCALING**: refer to page 16.

The path acceleration remains set until you change it with a subsequent path acceleration command. Accelerations outside the valid range are flagged as an error, with a message `*INVALID DATA-FIELD x`, where x is the field number. When an invalid acceleration is entered the previous acceleration value is retained.

If the path deceleration (PAD) command has not been entered, the path acceleration (PA) command will set the path deceleration rate. Once the path deceleration (PAD) command has been entered, the path acceleration (PA) command no longer affects path deceleration.

Example:
```
PV5                ; Set path velocity to 5 units/sec
PA50               ; Set path acceleration to 50 units/sec/sec
PAD100             ; Set path deceleration to 100 units/sec/sec
DEF prog1          ; Begin definition of path named prog1
PAXES1,2           ; Set axes 1 and 2 as the X and Y contouring axes
PAB0               ; Set to incremental coordinates
PLIN1,1            ; Specify X-Y endpoint position to create a 45 degree
                   ; angle line segment
END                ; End definition of path prog1
PCOMP prog1        ; Compile path prog1
PRUN prog1         ; Execute path prog1
```

PAA   Path Average Acceleration

Type	Motion (S-Curve); Motion (Linear Interpolated)	**Product**	**Rev**
Syntax	`<!>PAA<r>`	6K	5.0
Units	r = units/sec/sec (scalable by SCLD)		
Range	0.00001-39,999,998 (depending on the scaling factor)		
Default	10.00 (trapezoidal profiling is default, where PAA tracks PA)		
Response	PAA: *PAA10.0000		
See Also	DRES, PA, PAD, PADA, SCLD, SCALE		

The Path Average Acceleration (PAA) command allows you to specify the average acceleration for an S-curve path profile. S-curve profiling provides smoother motion control by reducing the rate of change in acceleration and deceleration; this accel/decel rate of change is known as *jerk*. S-curve profiling improves position tracking performance in linear interpolation applications (not contouring). S-curve profiling is <u>not</u> available for contouring applications. Refer to page 13 for details on S-curve profiling.

NOTE: Path scaling (`SCLD`) affects `PAA` the same as it does for `PA`. Refer to page 16 for details on scaling.

Example:
```
PV5              ; Set path velocity to 5 units/sec
PA50             ; Set path acceleration to 50 units/sec/sec
PAA40            ; Set path s-curve (average) acceleration to 40 units/sec/sec
PAD100           ; Set path deceleration to 100 units/sec/sec
PADA70           ; Set path s-curve (average) deceleration to 70 units/sec/sec
DEF prog1        ; Begin definition of path named prog1
D10,5,2,11       ; Set distance values, axes 1-4
GOL1111          ; Initiate linear interpolation motion
END              ; End definition of path prog1
```

PAB   Path Absolute

Type	Path Contouring	**Product**	**Rev**
Syntax	`<!>PAB<b>`	6K	5.0
Units	n/a		
Range	b = 0 (incremental) or 1 (absolute)		
Default	0		
Response	No response - Must be defining a path (DEF)		
See Also	PL, PLC, SCLD, PWC, SCALE		

The Path Absolute (`PAB`) command is used to indicate whether the subsequent segment endpoints are specified in either incremental (Ø) or absolute (1) coordinates. Segment endpoint position specifications may be either absolute with respect to the user-defined coordinate system, or incremental, relative to the start of each individual segment. At any point along a path definition, coordinates may be switched from incremental to absolute.

The absolute coordinate system may be either the *work* coordinate system or the *local* coordinate system (see `PL`).

PAD   Path Deceleration

Type	Path Contouring or Motion (Linear Interpolated)	**Product**	**Rev**
Syntax	`<!>PAD<r>`	6K	5.0
Units	r = units/sec/sec (scalable by SCLD)		
Range	0.00001-39,999,998 (depending on the scaling factor)		
Default	10.0000 (PAD tracks PA)		
Response	PAD: *PAD10.0000		
See Also	GOL, PA, PAA, PADA, SCLD, SCALE		

The Path Deceleration (`PAD`) command specifies the path deceleration to be used with linearly interpolated moves (`GOL`), and all contouring moves (`PLIN`, `PARCM`, `PARCOM`, `PARCOP`, `PARCP`). For both the linear interpolated and the contouring moves, the path deceleration refers to the deceleration experienced by the load as motion slows along the path. For linearly interpolated moves, the deceleration of each individual axis is dependent on the distance it contributes to the total path traveled by the load. In contouring paths, the deceleration of each individual axis is dependent on the direction of travel in the X-Y plane.

> **UNITS OF MEASURE** and **SCALING**: refer to page 16.

The path deceleration remains set until you change it with a subsequent path deceleration command. Decelerations outside the valid range are flagged as an error, with a message `*INVALID DATA-FIELD x`, where x is the field number. When an invalid deceleration is entered the previous deceleration value is retained.

If the path deceleration (`PAD`) command has not been entered, the path acceleration (`PA`) command will set the path deceleration rate. Once the path deceleration (`PAD`) command has been entered, the path acceleration (`PA`) command no longer affects path deceleration. If `PAD` is set to zero (`PADØ`), then the path deceleration will once again track whatever the `PA` command is set to.

Example: Refer to the path acceleration (`PA`) command example.

PADA   Path Average Deceleration

Type	Motion (S-Curve); Motion (Linear Interpolated)	**Product**	**Rev**
Syntax	`<!>PADA<r>`	6K	5.0
Units	r = units/sec/sec (scalable by SCLD)		
Range	0.00001-39,999,998 (depending on the scaling factor)		
Default	10.00 (PADA tracks PAA)		
Response	PADA: *PADA10.0000		
See Also	DRES, PA, PAA, PAD, SCLD, SCALE		

Use the Path Average Deceleration (PADA) command to specify the average deceleration for an S-curve path profile. S-curve profiling provides smoother motion control by reducing the rate of change in acceleration and deceleration; this accel/decel rate of change is known as *jerk*. S-curve profiling can improve position tracking performance in linear interpolation applications (not contouring). S-curve profiling is <u>not</u> available for contouring applications. Refer to page 13 for details on S-curve profiling.

NOTE: Path scaling (SCLD) affects PADA the same as it does for PAD. Refer to page 16 for details on scaling.

Example: Refer to the path average acceleration (PAA) command example.

[PANI]   Position of ANI Inputs

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	[ANI], ANIRNG, [FB], [CA], CMDDIR, PSET, SCALE, SCLD, SFB, TANI, TPANI, TFB		

This command is available only to servo axes, and only if you have an analog input SIM in an extended I/O brick.

The PANI operator is used to assign the analog input's position information to a variable, or to make a comparison against another value. The PANI value represents the analog input position after the affects of distance scaling (SCLD), offset (PSET), and commanded direction polarity (CMDDIR).

The TPANI and PANI commands are designed for applications in which analog input is scaled and/or used as position feedback. If you are using analog input to monitor an analog signal, the TANI and ANI commands would be more appropriate (TANI and ANI values are measured in volts and are unaffected by scaling, offset, or command direction polarity).

The PANI value is represented in analog-to-digital converter (ADC) units if scaling is disabled (SCALEØ). The ADC has a 12-bit resolution, giving a range of +2047 to -2048 counts when using the full ±10V range of the analog input (205 counts/volt). If scaling is enabled (SCALE1), an SCLD scale factor of 205 (the default value when analog input feedback is selected) allows units of volts to be used.

NOTE: If you change the voltage range of the analog input (with the ANIRNG command), the resolution of the PANI response will change accordingly. The default is ±10V (+2047 to -2048 counts).

Syntax: VARn=<B>PANI.i where "n" is the variable number, "<B>" is the number of the I/O brick, and "i" is I/O brick address where the analog input resides; or PANI can be used in an expression such as IF(1PANI.10=2.3). If no brick identifier (<B>) is provided, it defaults to 1. To understand the I/O brick addressing convention, refer to page 6.

Example:
```
SCLD205          ; Set distance scaling to accommodate values in volts
                 ; (205 counts/volt)
SCALE1           ; Enable scaling
DEL proge        ; Delete program called proge
DEF proge        ; Begin definition of program called proge
VAR4=1PANI.10    ; Position of the 2nd analog input on SIM2 of I/O brick 1
                 ; is assigned to variable 4
IF(1PANI.9<8.2)  ; If position of 1st analog input on SIM2 of I/O brick 1
                 ; is < 8.2 volts, do the commands between IF and NIF
TREV             ; Transfer revision level
NIF              ; End if statement
END              ; End definition of proge
```

PARCM   Radius Specified CCW Arc Segment

Type	Path Contouring	**Product**	**Rev**
Syntax	<!>PARCM<r>,<r>,<r>	6K	5.0
Units	r = units (scalable by the SCLD value)		
Range	0.00000 – ±999,999,999		
Default	n/a		
Response	No response – Must be defining a path (DEF)		
See Also	PARCP, PARCOM, PARCOP, PRTOL, SCLD, SCALE		

The Radius Specified CCW Arc Segment (PARCM) command is used to specify the endpoints and the radius of a counter-clockwise arc segment. The placement, length, radius of curvature, and orientation of the arc are completely specified by the endpoint and radius specifications of the arc segment and the endpoint of the previous segment (current position). The direction of rotation in the X-Y plane will be counter-clockwise.

A complete circle cannot be specified with a PARCM command, because the center is arbitrary. **Use the PARCOM command for circles.**

Command Syntax: PARCM<Xend>,<Yend>,<Radius>

Segment endpoint position specifications may be either absolute (PAB1) with respect to user defined segment start coordinates, or incremental (PAB0), relative to the start of each individual segment. The first two numbers following the PARCM command specify the X endpoint and the Y endpoint, respectively.

Radius specifications are signed values. A positive radius specifies an arc which is 180 degrees or less. A negative radius specifies an arc which is 180 degrees or more. The last number of the PARCM command specifies the radius.

> **UNITS OF MEASURE** and **SCALING**: refer to page 16 or to the SCLD description.

Example
```
PV5              ; Set path velocity to 5 units/sec
PA50             ; Set path acceleration to 50 units/sec/sec
PAD100           ; Set path deceleration to 100 units/sec/sec
PSET0,0          ; Set absolute position to 0,0
DEF prog1        ; Begin definition of path named prog1
PAXES1,2         ; Set axes 1 and 2 as the X and Y contouring axes
PAB0             ; Set to incremental coordinates
POUT1001         ; Output pattern during first arc: onboard outputs 1 & 4 are
                 ; on and outputs 2 & 3 are off
PARCM5,5,5       ; Specify incremental X-Y endpoint position and radius arc
                 ; <180 degrees for 1/4 circle counter-clockwise arc
POUT1100         ; Output pattern during second arc: onboard outputs 1 & 2 are
                 ; on and outputs 3 & 4 are off
PARCP5,-5,-5     ; Specify incremental X-Y endpoint position and radius arc
                 ; >180 degrees for 3/4 circle clockwise arc
END              ; End definition of path prog1
PCOMP prog1      ; Compile path prog1
PRUN prog1       ; Execute path prog1
OUT0000          ; Turn off the first four onboard outputs
```

PARCOM   Origin Specified CCW Arc Segment

Type	`Path Contouring`	**Product**	**Rev**
Syntax	`<!>PARCOM<r>,<r>,<r>,<r>`	6K	5.0
Units	`r = units (scalable with the SCLD value)`		
Range	`0.00000 – ±999,999,999`		
Default	`n/a`		
Response	`No response – Must be defining a path (DEF)`		
See Also	`PARCOP, PARCM, PARCP, PRTOL, SCLD, SCALE`		

The Origin Specified CCW Arc Segment (`PARCOM`) command is used to specify the coordinates necessary to create a counter-clockwise arc segment. The placement, length, radius of curvature, and orientation of the arc are completely specified by the endpoint and center specifications of the arc segment and the endpoint of the previous segment (current position). The direction of rotation in the X-Y plane will be counter-clockwise.

Command Syntax: `PARCOM<Xend>,<Yend>,<Xcenter>,<Ycenter>`

Segment endpoint position specifications may be either absolute (`PAB1`) with respect to user defined segment start coordinates, or incremental (`PABØ`), relative to the start of each individual segment. The first two numbers following the `PARCOM` command specify the X endpoint and the Y endpoint, respectively.

Center position specifications are always incremental, relative to the start of the arc segment. The last two numbers following the `PARCOM` command specify the X center point and Y center point coordinates, respectively.

UNITS OF MEASURE and **SCALING**: refer to page 16 or to the `SCLD` description.

Example:
```
PV5                  ; Set path velocity to 5 units/sec
PA50                 ; Set path acceleration to 50 units/sec/sec
PAD100               ; Set path deceleration to 100 units/sec/sec
PSET0,0              ; Set absolute position to 0,0
DEF prog1            ; Begin definition of path named prog1
PAXES1,2             ; Set axes 1 and 2 as the X and Y contouring axes
PAB0                 ; Set to incremental coordinates
POUT1001             ; Output pattern during first arc: onboard outputs 1 & 4 are
                     ; on and outputs 2 & 3 are off
PARCOM5,5,0,5        ; Specify incremental X-Y endpoint position and X-Y center
                     ; position for quarter circle counter-clockwise arc
POUT1100             ; Output pattern during second arc: onboard outputs 1 & 2 are
                     ; on and outputs 3 & 4 are off
PARCOP0,0,5,0        ; Specify incremental X-Y endpoint position and X-Y center
                     ; position for full circle clockwise arc
END                  ; End definition of path prog1
PCOMP prog1          ; Compile path prog1
PRUN prog1           ; Execute path prog1
OUT0000              ; Turn off the first four onboard outputs
```

PARCOP   Origin Specified CW Arc Segment

Type	Path Contouring	**Product**	**Rev**
Syntax	`<!>PARCOP<r>,<r>,<r>,<r>`	6K	5.0
Units	`r = units (scalable by the SCLD value)`		
Range	`0.00000 – ±999,999,999`		
Default	`n/a`		
Response	`No response – Must be defining a path (DEF)`		
See Also	`PARCOM, PARCM, PARCP, PRTOL, SCLD, SCALE`		

The Origin Specified CW Arc Segment (`PARCOP`) command is used to specify the coordinates necessary to create a clockwise arc segment. The placement, length, radius of curvature, and orientation of the arc are completely specified by the endpoint and center specifications of the arc segment and the endpoint of the previous segment (current position). The direction of rotation in the X-Y plane will be clockwise.

Command Syntax: `PARCOP<Xend>,<Yend>,<Xcenter>,<Ycenter>`

Segment endpoint position specifications may be either absolute (`PAB1`) with respect to user defined segment start coordinates, or incremental (`PAB0`), relative to the start of each individual segment. The first two numbers following the `PARCOP` command specify the X endpoint and the Y endpoint, respectively.

Center position specifications are always incremental, relative to the start of the arc segment. The last two numbers following the `PARCOP` command specify the X center point and Y center point coordinates, respectively.

> **UNITS OF MEASURE** and **SCALING**: refer to page 16 or to the `SCLD` description.

Example: Refer to the `PARCOM` command example.

PARCP   Radius Specified CW Arc Segment

Type	Path Contouring	**Product**	**Rev**
Syntax	`<!>PARCP<r>,<r>,<r>`	6K	5.0
Units	`r = units (scalable by the SCLD value)`		
Range	`0.00000 – ±999,999,999`		
Default	`n/a`		
Response	`No response – Must be defining a path (DEF)`		
See Also	`PARCM, PARCOM, PARCOP, PRTOL, SCLD, SCALE`		

The Radius Specified CW Arc Segment (`PARCP`) command is used to specify the endpoints and the radius of a clockwise arc segment. The placement, length, radius of curvature, and orientation of the arc are completely specified by the endpoint and radius specifications of the arc segment and the endpoint of the previous segment (current position). The direction of rotation in the X-Y plane will be clockwise.

*A complete circle cannot be specified with a `PARCP` command, because the center is arbitrary. **Use the `PARCOP` command for circles.***

Command Syntax: `PARCP<Xend>,<Yend>,<Radius>`

Segment endpoint position specifications may be either absolute (`PAB1`)with respect to user defined segment start coordinates, or incremental (`PAB0`), relative to the start of each individual segment. The first two numbers following the `PARCP` command specify the X endpoint and the Y endpoint, respectively.

Radius specifications are signed values. A positive radius specifies an arc which is 180 degrees or less. A negative radius specifies an arc which is 180 degrees or more. The last number of the `PARCP` command specifies the radius.

> **UNITS OF MEASURE** and **SCALING**: refer to page 16 or to the `SCLD` description.

Example: Refer to the `PARCM` command example.

PAXES   Set Contouring Axes

		Product	Rev
Type	`Path Contouring`	6K	5.0
Syntax	`<!>PAXES<i>,<i>,<i>,<i>`		
Units	Each `<i>`: X axis, Y axis, Tangent axis, Proportional axis		
Range	`i = 1-8 (product dependent)`		
Default	`1,2,0,0`		
Response	`No response - Must be defining a path (DEF)`		
See Also	`DEF, DRES, END, ERES, PCOMP, PPRO, PRUN, SCLD, TSKAX`		

The Set Contouring Axes (PAXES) command defines the axes to be used in the current path definition (syntax: `PAXES<Xaxis>,<Yaxis>,<Tangent>,<Proportional>`). The X and Y axes must be specified, but the Tangent and Proportional axes are optional.

If no axis number is specified for the Tangent or Proportional axes, it signifies that the Tangent or Proportional axes are not included in that path definition. The axis specification for the entire path is done with this command. The PAXES command should be given prior to any contour segments.

NOTES

- For products that control only 2 axes of motion, the Tangent and Proportional axes are not available.
- When using scaling (SCALE1), the units used for path distance, acceleration, and velocity is determined by the SCLD value. For example, suppose you have 2 servo axes (axes 1 & 2) involved in contouring, both axes use encoder feedback with a resolution of 4000 counts/rev, axis 1 uses a 10:1 (10 turns per inch) leadscrew and axis 2 uses a 5:1 (5 turns per inch) lead screw, and you want to program in inches. For this application you would use the SCLD40000,20000 command to establish path motion units in inches: distance is inches, acceleration is inches/sec/sec, and velocity is inches/sec.
- When <u>not</u> using scaling (SCALE0), path motion units are based on the resolution (DRES for steppers, ERES for servos) of axis 1. If multi-tasking is used, path motion units are based on the resolution of the first (lowest number) axis associated with the task (TSKAX).

Example: (see `PCOMP`)

[PC]   Position Commanded

		Product	Rev
Type	`Assignment or Comparison`	6K	5.0
Syntax	See below		
Units	`n/a`		
Range	`n/a`		
Default	`n/a`		
Response	`n/a`		
See Also	`ERES, [ FB ], GOWHEN, [ PCC ], [ PE ], [ PER ], PSET, SCALE,` `SCLD, SMPER, TAS, TFB, TPC, TPCC, TPE, TPER`		

Use PC operator to assign the current *commanded position* (scalable by SCLD) of each axis to a variable, or to make a comparison against another value. If you issue a PSET command, the commanded position value will be offset by the PSET command value.

Servo Axes: The PC value is measured in encoder or analog input (ANI) counts. The commanded position (PC) and the actual position (FB) are used in the control algorithm to calculate the position error (PC - FB = PER) and thereby determine the corrective control signal.

Stepper Axes: The PC value is measured in commanded counts ("motor counts").

UNITS OF MEASURE and **SCALING**: refer to page 16.

Syntax: `VARn=aPC` where "n" is the variable number, and "a" is the axis, or PC can be used in an expression such as `IF(1PC>5Ø)`. The PC command must be used with an axis specifier or it will default to axis 1 (e.g., 1PC, 2PC, etc.).

Example:
```
VAR1=1PC        ; Commanded position for axis 1 is assigned to variable 1
IF(2PC<50)      ; If the commanded position for axis 2 is <50, do the IF statement
VAR2=2PC+500    ; Commanded position for axis 2 plus 500 is assigned to variable 2
NIF             ; End IF statement
```

[PCC]   **Captured Commanded Position**

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	`CMDDIR, ENCCNT, INFNC, [ PC ], [ PCMS ], PSET, SCALE, SCLD,`		
	`SFB, [ TRIG ], TRGLOT, TPC, TPCC, TTRIG`		

Use the `PCC` operator to assign the captured commanded position of a specific axis to a variable, or to make a comparison against another value.

> **Syntax**: `VARn=aPCCc` where "n" is the variable number, "a" is the axis, and "c" designates trigger A or B for the axis, or M for the **MASTER TRIG** input (see table below); or `PCC` can be used in an expression such as `IF(1PCCB>2345Ø)`. The `PCC` operator must be used with an axis specifier or it will default to axis 1 (e.g., `1PCCA`, `2PCCB`, `5PCCM`, etc.).

Trigger Input (Axis 1-4 "TRIGGERS/OUTPUTS" connector) *		Dedicated Axis	PCC Syntax	Trigger Input (Axis 5-8 "TRIGGERS/OUTPUTS" connector) *		Dedicated Axis	PCC Syntax
Pin 23,	Trigger 1A	1	`1PCCA`	Pin 23,	Trigger 5A	5	`5PCCA`
Pin 21,	Trigger 1B	1	`1PCCB`	Pin 21,	Trigger 5B	5	`5PCCB`
Pin 19,	Trigger 2A	2	`2PCCA`	Pin 19,	Trigger 6A	6	`6PCCA`
Pin 17,	Trigger 2B	2	`2PCCB`	Pin 17,	Trigger 6B	6	`6PCCB`
Pin 15,	Trigger 3A	3	`3PCCA`	Pin 15,	Trigger 7A	7	`7PCCA`
Pin 13,	Trigger 3B	3	`3PCCB`	Pin 13,	Trigger 7B	7	`7PCCB`
Pin 11,	Trigger 4A	4	`4PCCA`	Pin 11,	Trigger 8A	8	`8PCCA`
Pin 9,	Trigger 4B	4	`4PCCB`	Pin 9,	Trigger 8B	8	`8PCCB`

* The number of trigger inputs available varies by product (refer to your product's *Installation Guide*).

To use an axis position captured with the MASTER TRIG input, use `aPCCM`, where "a" can be any axis number.

About Position Capture: The commanded position can be captured only by a trigger input that is defined as "trigger interrupt" input with the `INFNCi-H` command (see `INFNC` for details). Each trigger input, when configured as a "trigger interrupt" input, is dedicated to capture the position of a specific axis (see table above). When a "trigger interrupt" input is activated, the commanded position of the dedicated axis is captured and the position is available through the use of the `PCC` operator and the `TPCC` display command.

Note for Stepper Axes: By default, stepper axes capture only the commanded position. However, if the axis has Encoder Capture Mode enabled with the `ENCCNT` command, only the encoder position is captured.

Position Capture Status, Longevity of Captured Position: Use the `TTRIG` and `TRIG` commands to ascertain if a trigger interrupt input has been activated. `TTRIG` displays the status as a binary report, and `TRIG` is an assignment/comparison operator for using the status information in a conditional expression (e.g., in an `IF` statement). Once the captured commanded position value is assigned/compared with the `PCC` operator, the `TTRIG`/`TRIG` status bit for that trigger input is cleared; but the position information remains available until it is overwritten by a subsequent position capture from the same trigger input.

Position Capture Accuracy: The commanded position capture accuracy is ±1 count.

Scaling and Position Offset: If scaling is enabled (`SCALE1`), the commanded position is scaled by the distance scaling factor (`SCLD`). If scaling is not enabled (`SCALEØ`), the value assigned will be actual commanded counts. If you issue a `PSET` (establish absolute position reference) command, any previously captured commanded positions will be offset by the `PSET` command value.

Example:
```
INFNC1-H        ; Assign trigger input 1A as trigger interrupt input for axis 1
INFNC3-H        ; Assign trigger input 2A as trigger interrupt input for axis 2
VAR1=1PCCA      ; Assign captured commanded position of axis 1 to variable 1
                ; (position was captured when trigger input 1A became active)
IF(2PCCA<40)    ; If the captured commanded position on axis 2
                ; (captured when trigger input 2A became active) is
                ; less than 40, do the IF statement
VAR2=1PCCA+10   ; Add 10 to the captured commanded position on axis 1
                ; (captured when trigger input 1A became active) and
                ; assign the sum to variable #2
NIF             ; End IF statement
```

[PCE]   Position of Captured Encoder

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	CMDDIR, ENCCNT, ENCPOL, INFNC, [PCMS], [PE], PSET, SCALE, SCLD, SFB, TPCE, [TRIG], TRGLOT, TTRIG		

Use the PCE operator to assign the captured encoder position of a specific axis to a variable, or to make a comparison against another value.

> **Syntax**: VARn=aPCEc where "n" is the variable number, "a" is the axis, and "c" designates trigger A or B for the axis, or M for the **MASTER TRIG** input (see table below); or PCE can be used in an expression such as IF(1PCEB>2345Ø). The PCE operator must be used with an axis specifier or it will default to axis 1 (e.g., 1PCEA, 2PCEB, 5PCEM, etc.).

Trigger Input (Axis 1-4 "TRIGGERS/OUTPUTS" connector) *		Dedicated Axis	PCE Syntax	Trigger Input (Axis 5-8 "TRIGGERS/OUTPUTS" connector) *		Dedicated Axis	PCE Syntax
Pin 23,	Trigger 1A	1	1PCEA	Pin 23,	Trigger 5A	5	5PCEA
Pin 21,	Trigger 1B	1	1PCEB	Pin 21,	Trigger 5B	5	5PCEB
Pin 19,	Trigger 2A	2	2PCEA	Pin 19,	Trigger 6A	6	6PCEA
Pin 17,	Trigger 2B	2	2PCEB	Pin 17,	Trigger 6B	6	6PCEB
Pin 15,	Trigger 3A	3	3PCEA	Pin 15,	Trigger 7A	7	7PCEA
Pin 13,	Trigger 3B	3	3PCEB	Pin 13,	Trigger 7B	7	7PCEB
Pin 11,	Trigger 4A	4	4PCEA	Pin 11,	Trigger 8A	8	8PCEA
Pin 9,	Trigger 4B	4	4PCEB	Pin 9,	Trigger 8B	8	8PCEB

* The number of trigger inputs available varies by product (refer to your product's *Installation Guide*).

To use an axis position captured with the MASTER TRIG input, use aPCEM, where "a" can be any axis number.

About Position Capture: The encoder position can be captured only by a trigger input that is defined as "trigger interrupt" input with the INFNCi-H command (see INFNC command). Each trigger input, when configured as a "trigger interrupt" input, is dedicated to capture the position of a specific axis (see table above). When a "trigger interrupt" input is activated, the encoder position of the dedicated axis is captured and the position is available through the use of the PCE operator and the TPCE display command. **Stepper Axes**: By default, stepper axes capture only the commanded position. To capture the encoder position, the axis must be in the Encoder Capture Mode (see ENCCNT command).

Position Capture Status, Longevity of Captured Position: Use the TTRIG and TRIG commands to ascertain if a trigger interrupt input has been activated. TTRIG displays the status as a binary report, and TRIG is an assignment/comparison operator for using the status information in a conditional expression (e.g., in an IF statement). Once the captured encoder position value is assigned/compared with the PCE operator, the TTRIG/TRIG status bit for that trigger input is cleared; but the position information remains available until it is overwritten by a subsequent position capture from the same trigger input.

Position Capture Accuracy: The encoder position capture accuracy is ±1 encoder count.

Scaling and Position Offset: If scaling is enabled (SCALE1), the encoder position is scaled by the distance scaling factor (SCLD). If scaling is not enabled (SCALEØ), the value assigned will be actual encoder counts. If you issue a PSET (establish absolute position reference) command, any previously captured encoder positions will be offset by the PSET command value.

Example:
```
INFNC1-H        ; Assign trigger input 1A as trigger interrupt input for axis 1
INFNC3-H        ; Assign trigger input 2A as trigger interrupt input for axis 2
VAR1=1PCEA      ; Assign captured encoder position of axis 1 to variable 1
                ; (position was captured when trigger input 1A became active)
IF(2PCEA<4000)  ; If the captured encoder position on axis 2
                ; (captured when trigger input 2A became active) is
                ; less than 4000, do the IF statement
VAR2=1PCEA+10   ; Add 10 to the captured encoder position on axis 1
                ; (captured when trigger input 1A became active) and
                ; assign the sum to variable #2
NIF             ; End IF statement
```

[PCME]   Position of Captured Master Encoder

			Product	Rev
Type	Assignment or Comparison		6K	5.0
Syntax	See below			
Units	n/a			
Range	n/a			
Default	n/a			
Response	n/a			
See Also	INFNC, MEPOL, MESND, [PME], [PCMS], PMECLR, PMESET, TPCME, TPME, TPCMS			

Use the PCME operator to assign the captured master encoder position to a variable, or to make a comparison against another value. The master encoder is connected to the connector labeled "Master Encoder."

> **Syntax**: VARn=PCME where "n" is the variable number; or PCME can be used in an expression such as IF(PCME>2345Ø).

About Position Capture: The master encoder position can be captured only by the Master Trigger input (labeled "MASTER TRIG"), and only when that input is defined as a "trigger interrupt" input with the INFNC17-H command (see INFNC command). When the "trigger interrupt" input is activated (active edge), the master encoder position is captured and the position is available through the use of the PCME operator and the TPCME display command.

Position Capture Status, Longevity of Captured Position: Use the TTRIG and TRIG commands to ascertain if a trigger interrupt input has been activated. TTRIG displays the status as a binary report, and TRIG is an assignment/comparison operator for using the status information in a conditional expression (e.g., in an IF statement). Once the captured master encoder position value is assigned/compared with the PCME operator, TTRIG/TRIG status bit #17 is cleared; but the position information remains available until it is overwritten by a subsequent position capture from the master trigger input.

Position Capture Accuracy: The master encoder position capture accuracy is ±1 encoder count.

Scaling and Position Offset: The PCME value is always in master encoder counts; it is never scaled. If you issue a PMESET (establish absolute position reference) command, any previously captured master encoder positions will be offset by the PMESET command value.

Example:
```
INFNC17-H          ; Assign master trigger as trigger interrupt input for the
                   ; master encoder
VAR1=PCME          ; Assign captured master encoder position to variable 1
                   ; (position was captured when master trigger became active)
IF(PCME<4000)      ; If the captured master encoder position
                   ; (captured when master trigger input became active) is
                   ; less than 4000, do the IF statement
VAR2=PCME+10       ; Add 10 to the captured master encoder position
                   ; (captured when master trigger input became active) and
                   ; assign the sum to variable #2
NIF                ; End IF statement
```

[PCMS]   Captured Master Cycle Position

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	CMDDIR, ENCCNT, ENCPOL, FOLMAS, INFNC, [PCC], [PCE], [PE], PSET, SCALE, SCLMAS, SFB, TPCMS, [TRIG], TRGLOT, TTRIG		

Use the PCMS operator to assign the captured master cycle position for a specific follower axis to a variable, or to make a comparison against another value.

PCMS (like PMAS) is unique among position assignment variables, because its value rolls over to zero each time the entire master cycle length (FMCLEN) has been traveled. Thus, the captured PCMS value is essentially a snap-shot of the position relative to the master cycle at the time of the capture.

The master must be assigned first (FOLMAS command) before this operator will be useful.

> **Syntax**: VARn=aPCMSc where "n" is the variable number, "a" is the axis, and "c" designates trigger A or B for the axis, or M for the **MASTER TRIG** input (see table below); or PCMS can be used in an expression such as IF(1PCMSB>2311). The PCMS operator must be used with an axis specifier or it will default to axis 1 (e.g., 1PCMSA, 2PCMSB, 5PCMSM, etc.).

Trigger Input (Axis 1-4 "TRIGGERS/OUTPUTS" connector) *		Dedicated Axis	PCMS Syntax	Trigger Input (Axis 5-8 "TRIGGERS/OUTPUTS" connector) *		Dedicated Axis	PCMS Syntax
Pin 23,	Trigger 1A	1	1PCMSA	Pin 23,	Trigger 5A	5	5PCMSA
Pin 21,	Trigger 1B	1	1PCMSB	Pin 21,	Trigger 5B	5	5PCMSB
Pin 19,	Trigger 2A	2	2PCMSA	Pin 19,	Trigger 6A	6	6PCMSA
Pin 17,	Trigger 2B	2	2PCMSB	Pin 17,	Trigger 6B	6	6PCMSB
Pin 15,	Trigger 3A	3	3PCMSA	Pin 15,	Trigger 7A	7	7PCMSA
Pin 13,	Trigger 3B	3	3PCMSB	Pin 13,	Trigger 7B	7	7PCMSB
Pin 11,	Trigger 4A	4	4PCMSA	Pin 11,	Trigger 8A	8	8PCMSA
Pin 9,	Trigger 4B	4	4PCMSB	Pin 9,	Trigger 8B	8	8PCMSB

* The number of trigger inputs available varies by product (refer to your product's *Installation Guide*).

To use a position captured with the MASTER TRIG input, use aPCMSM, where "a" can be any axis number.

About Position Capture: The master cycle position can be captured only by a trigger input that is defined as "trigger interrupt" input with the INFNCi-H command (see INFNC command). Each trigger input, when configured as a "trigger interrupt" input, is dedicated to capture the position of a specific axis (see table above). When a "trigger interrupt" input is activated, the master cycle position of the dedicated axis is captured and the position is available through the use of the PCMS operator and the TPCMS display command.

Position Capture Status, Longevity of Captured Position: Use the TTRIG and TRIG commands to ascertain if a trigger interrupt input has been activated. TTRIG displays the status as a binary report, and TRIG is an assignment/comparison operator for using the status information in a conditional expression (e.g., in an IF statement). Once the captured master cycle position value is assigned/compared with the PCMS operator, the TTRIG/TRIG status bit for that trigger input is cleared; but the position information remains available until it is overwritten by a subsequent position capture from the same trigger input.

Position Capture Accuracy: The master cycle position is interpolated; the capture accuracy is 50 µs multiplied by the velocity of the axis at the time the trigger input was activated.

Scaling and Position Offset: If scaling is enabled (SCALE1), the master source position is scaled by the distance scaling factor (SCLMAS). If scaling is not enabled (SCALE0), the value assigned will be actual counts from the commanded or encoder master source as selected with the FOLMAS command. If you issue a PSET (establish absolute position reference) command, any previously captured master cycle positions will be offset by the PSET command value.

PCOMP   Compile a Profile or Program

			Product	**Rev**
Type	Compiled Motion; Path Contouring; PLC Program		6K	5.0
Syntax	`<!>PCOMP<t>`			
Units	t = text (name of program/path)			
Range	Text name of 6 characters or less			
Default	n/a			
Response	n/a			
See Also	DEF, DRES, END, GOBUF, GOWHEN, MEMORY, PA, PAA, PAD, PADA, PAB, PARCOM, PARCOP, PARCM, PARCP, PAXES, PEXE, PLOOP, PL, PLC, PLCP, PLIN, PLN, POUTn, PRUN, SCLD, PUCOMP, PULSE, SCANP, [SEG], [SS], TDIR, TMEM, TRGFN, TSEG, TSS			

Use the `PCOMP` command to compile multi-axis contours, compiled (`GOBUF`) profiles for individual axes, and compiled `PLCP` programs for PLC Scan Mode. (For additional detail on contouring and compiled motion, refer to the Custom Profiling chapter in the *Programmer's Guide*.)

"Programs" vs. "Compiled Profiles & Programs":

- Programs are defined with the `DEF` and `END` commands, as demonstrated in the Program Development Scenario in the *Programmer's Guide*.
- Compiled Profiles are defined like programs (using the `DEF` and `END` commands), but are compiled with the `PCOMP` command and executed with the `PRUN` command. A compiled profile could be a multi-axis contour (a series of arcs and lines), an individual axis profile (a series of `GOBUF` commands), or a compound profile (combination of multi-axis contours and individual axis profiles).
- Compiled PLC programs are defined with `DEF PLCPi` and `END`, compiled with `PCOMP`, and are normally executed in the PLC Scan Mode with the `SCANP`.

Compiling and Storing Compiled Paths & Programs:

Your controller's memory has two partitions: one for storing programs ("program" memory) and one for storing profiles & program segments compiled with the `PCOMP` command ("compiled" memory). The allocation of memory to these two areas is controlled with the `MEMORY` command.

Programs intended to be compiled are stored in program memory. After they are compiled with the `PCOMP` command, they remain in program memory and the segments (see segment command list below) from the compiled profile are stored in compiled memory.

- Contouring segments: PARCM, PARCOM, PARCOP, PARCP, PLIN
- Compiled Motion segments: GOBUF, PLOOP, GOWHEN, TRGFN, POUTA, POUTB, POUTC, POUTD
- PLC Program segments: IF, ELSE, NIF, L, LN, OUT, EXE, PEXE, VARI, VARB

The `TDIR` command uses "COMPILED AS A PATH" to denote the programs compiled as a compiled profile, and "COMPILED AS A PLC PROGRAM" to denote the programs compiled as a PLC programs. `TDIR` also reports the amount of program storage available, as does the `TSEG` command. System status bit #29 indicates that compiled memory is 75% full, and system status bit #30 indicates that compiled memory is completely full. (Use `TSSF`, `TSS` and `SS` to work with system status bits.)

If a compile (`PCOMP`) fails, system status bit #31 (see `TSSF`, `TSS` and `SS`) will be set. This status bit is cleared on power-up, reset, or after a successful compile. Possible causes for a failed compile are:

- Errors in profile design (e.g., change direction while at non-zero velocity; distance and velocity equate to < 1 count/system update; preset move profile ends in non-zero velocity).
- Profile will cause a Following error (see `TFSF`, `TFS` and `[FS]` commands).
- Out of memory (see system status bit #30).
- Axis already in motion at the time of a `PCOMP` command.
- Loop programming errors (e.g., no matching `PLOOP` or `PLN`; more than four embedded `PLOOP`/`END` loops).
- `PLCP` program contains invalid commands or command parameters.

Conditions That Require a Re-Compile (Contouring and Compiled Motion only):

- If it is desired to change a compiled path's velocity, acceleration, or deceleration, the values must be changed and then the path must be re-compiled.

- If the scaling factors are changed, the program must be downloaded again.

- Compiled Motion ONLY: After compiling (PCOMP) and running (PRUN) a compiled profile, the profile segments will be deleted from compiled memory if you cycle power or issue a RESET command.

COMPILED MOTION

When using compiled loops (PLOOP and PLN), the last segment within the loop must end at zero velocity or there must be a final GOBUF segment placed outside the loop. Otherwise an error will result when the profile is compiled. The error is "ERROR: MOTION ENDS IN NON-ZERO VELOCITY-AXIS n".

PLC PROGRAM EXAMPLE: see PLCP command description.

CONTOURING EXAMPLE
```
DEF prog1       ; Begin definition of program named prog1
PAXES1,2,3,4    ; Set axes 1, 2, 3, & 4 as the X, Y, Tangent, &
                ; Proportional axes, respectively
PPRO2.25        ; Proportional axis path ratio = 2.25
; **********************************************
; * Put                                        *
; * MULTIPLE MOTION SEGMENT DEFINITIONS        *
; * Here                                       *
; **********************************************
END             ; End definition of path prog1
PCOMP prog1     ; Compile path prog1
PRUN prog1      ; Execute path prog1
```

COMPILED MOTION EXAMPLE (see profile below)
```
DEF prog2       ; Begin definition of program named prog2
A10,10          ; Set A, V, and D values for axes 1 and 2
V2,2
D2000,2000
GOBUF11         ; First segment of motion for axes 1 and 2
V4,4            ; New A,V, and D values
AD50,50
D1000,1000
GOBUF11         ; Second segment
END             ; End definition of prog2
PCOMP prog2     ; Compile prog2
PRUN prog2      ; Execute prog2
```

velocity

[PE] Position of Encoder

		Product	Rev
Type	Assignment or Comparison	6K	5.0
Syntax	See below		
Units	Encoder counts, or scaled by SCLD		
Range	n/a		
Default	n/a		
Response	n/a		
See Also	CMDDIR, ENCCNT, ENCPOL, ENCSND, [FB], GOWHEN, INFNC, [PC], [PCE], [PER], PESET, PSET, SCALE, SCLD, SFB, TFB, TPE		

The Position of Encoder (PE) operator is used to assign one of the encoder register values to a variable, or to make a comparison against another value. If the encoder has been configured to receive step and direction input (ENCSND), the PE operator will report the position as counted from the step and direction signal.

Stepper axes: If the ENCCNT1 mode is enabled PE reports the encoder position, but in ENCCNT0 mode (the factory default setting) the PE report represents the commanded position.

> **UNITS OF MEASURE** and **SCALING**: refer to page 16 or to the SCLD command.

If you issue a PSET command, the encoder position value will be offset by the PSET command value. If you are using a stepper axis in the ENCCNT1 mode, use the PESET command instead.

Syntax: VARn=aPE where "n" is the variable number, and "a" is the axis, or PE can be used in an expression such as IF(1PE>2345Ø). The PE command must be used with an axis specifier or it will default to axis 1 (e.g., 1PE, 2PE, etc.).

Example:
```
VAR1=1PE          ; Encoder position for axis 1 is assigned to variable 1
IF(2PE<4000)      ; If the encoder count for axis 2 is less than 4000,
                  ; do the IF statement
VAR2=3PE+4000     ; Encoder position for axis 3 plus 4000 is assigned
                  ; to variable 2
NIF               ; End IF statement
```

[PER] Position Error

		Product	Rev
Type	Assignment or Comparison	6K	5.0
Syntax	See below		
Units	n/a	(applicable to servo axes only)	
Range	n/a		
Default	n/a		
Response	n/a		
See Also	CMDDIR, DRES, ENCPOL, ERES, SCLD, SFB, SMPER, TAS, TPER, TPE, TPC		

The Position Error (PER) operator is used to assign the current position error of each axis to a variable, or to make a comparison against another value. The value assigned to the variable or the value against which the comparison is made is measured in feedback device counts and is scaled by the distance scaling factor (SCLD), if scaling is enabled with the SCALE1 command.

The position error is the difference between the commanded position and the actual position read by the feedback device. This error is calculated every sample period and can be displayed at any time using the TPER command.

Syntax: VARn=aPER where "n" is the variable number, and "a" is the axis, or PER can be used in an expression such as IF(1PER>5Ø). The PER command must be used with an axis specifier or it will default to axis 1 (e.g., 1PER, 2PER, etc.).

Example:
```
VAR1=1PER         ; Position error for axis 1 is assigned to variable 1
IF(2PER>2000)     ; If the position error for axis 2 is >2000 encoder counts,
                  ; do the IF statement (enable output #4)
OUTXXX1           ; Enable onboard output #4
NIF               ; End IF statement
```

PESET   Encoder Absolute Position Reference - Stepper Axes

Type	`Motion`	**Product**	**Rev**
Syntax	`<!><@>PESET<r>,<r>,<r>,<r>,<r>,<r>,<r>,<r>`	6K	5.0
Units	`r = units (absolute position of encoder)`		
Range	`±999,999,999.99999`		(applicable to stepper
Default	`n/a`		axes only)
Response	`n/a`		
See Also	AXSDEF, ENCCNT, ENCPOL, INFNC, [PCE], [PE], PMESET, PSET, SCALE, SCLD, TPCE, TPE		

Use the `PESET` command to offset the current absolute encoder position to establish an *absolute position reference* for the encoder reports (`TPE`, `PE`, `TPCE`, `PCE`). **NOTE**: PESET can only be used for axes that are defined as stepper axes with the `AXSDEF` command. All `PESET` values entered are in encoder steps, scalable by the `SCLD` value if scaling is enabled.

NOTE: If you issue a `PESET` command, any previously captured encoder positions (`INFNCi-H` or `LIMFNCi-H` function) will be offset by the `PESET` value.

Example:
```
AXSDEF0000          ; Define axes 1-4 as stepper axes
ENCCNT1111          ; Place axes 1-4 in the encoder count referencing mode
PESET0,0,0,1000     ; Set absolute position on axes 1, 2, and 3 to zero,
                    ; and axis 4 to 1000 units
TPE                 ; Display the new positions. The new encoder position
                    ; report should be: *TPE0,0,0,1000
```

PEXE   Execute a Compiled Program

Type	`PLC Mode Compiled Program Execution`	**Product**	**Rev**
Syntax	`i%PEXEt`	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	EXE, GOBUF, PCOMP, PLCP, SCANP		

Use the `PEXE` command to start a compiled `PLCP` program, compiled contouring path, or compiled `GOBUF` profile from within a compiled `PLCP` program. The `PEXE` command specifies the name of the compiled program, and the task in which it will be launched. The program named in the `PEXE` command need not be defined or compiled at the time the `PLCP` program is compiled; however, the program must be defined and compiled 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 compiled program. Note, however, that the `PEXE` program cannot be executed in the Task Supervisor (task 0).

The `PLCP` program will ignore the `PEXE` command if a currently running program is detected within the specified task; therefore, the `PEXE` 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 `PEXE` command will not interrupt a currently running program, nor will it interrupt a `WAIT` or `T` command. Also, if launching a compiled contouring path or `GOBUF` profile, the `PEXE` will not interrupt motion already in progress.

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). A `PEXE` 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 `PEXE` command to complete before the same `PEXE` command is issued again. As stated, the `PLCP` program will ignore the `PEXE` command if a currently running program is detected or motion is in progress on the participating axes, so timing must be considered when launching programs with the `PEXE` command.

To execute a non-compiled program from within a compiled `PLCP` program, use the `EXE` command.

Example:
```
DEF PLCP1           ; Define PLC program PLCP1
1%PEXE PLCP2        ; Launch compiled program PLCP2 in task 1
END
DEF PLCP2           ; Define PLC program PLCP2
OUT(VARB1)          ; Modify outputs
END
```

```
PCOMP PLCP1         ; Compile PLCP1
PCOMP PLCP2         ; Compile PLCP2
SCANP PLCP1         ; Scan with program PLCP1
VARB1=h0000         ; Set VARB1
TOUT                ; Check outputs (response is *TOUT0000_0000_0000_0000)
VARB1=b1010         ; Reassign VARB1
TOUT                ; Check outputs again (response is *TOUT1010_0000_0000_0000)
```

[PI]   PI (π)

Type	Operator (Trigonometric)	**Product**	**Rev**
Syntax	See examples below	6K	5.0
Units	n/a		
Range	n/a		
Default	n/a		
Response	n/a		
See Also	[=], [+], [-], [*], [/], [&], [	], [^], [~], [ATAN], [COS], IF, [SIN], [SQRT], [TAN], VAR		

The (PI) command is assigned the value 3.14159265. There are 2π radians in 360°. This command is useful for doing trigonometric functions in radian units (RADIAN command).

Example:
```
VAR1=PI             ; 3.14159265 is assigned to variable 1
VAR2=2 * PI         ; 2 pi is assigned to variable 2
```

PL   Define Path Local Mode

Type	Path Contouring	**Product**	**Rev**
Syntax	<!>PL<b>	6K	5.0
Units	n/a		
Range	b = 0 (work coordinates) or 1 (local coordinates)		
Default	0		
Response	No response – Must be defining a path (DEF)		
See Also	PAB, PLC, PWC		

The Define Path Local Mode (PL) command is used to specify the use of either the Local coordinate system or the Work coordinate system. Endpoints are allowed to be specified as absolute positions, and these positions may either be in the Work or the Local coordinate system. Programming may switch between Local and Work coordinates before any segment or group of segments.

When switching to Local coordinates, the starting coordinates of the next segment in the Local coordinate system must be specified with the PLC command before the PL1 command is issued.

When using the Work coordinate system (PLØ), the starting coordinates of the next segment in the Work coordinate system may be specified with the PWC command for the purpose of shifting the Work coordinate system. If the PWC command is not given, the previous Work coordinate system is used.

Example:
```
PV5             ; Set path velocity to 5 units/sec
PA50            ; Set path acceleration to 50 units/sec/sec
PAD100          ; Set path deceleration to 100 units/sec/sec
DEF prog1       ; Begin definition of path named prog1
PAXES1,2        ; Set axes 1 and 2 as the X and Y contouring axes
PAB1            ; Set to absolute coordinates
PWC0,0          ; Specify X and Y data, work coordinates
PL0             ; Specify work coordinate system
PLIN1,1         ; Specify X-Y endpoint position to create a 45 degree angle line segment
PLC0,0          ; Specify X and Y data, local coordinates
PL1             ; Specify local coordinate system
PARCOP0,0,5,0   ; Specify incremental X-Y endpoint position and X-Y center
                ; position for full circle clockwise arc
PLIN0,11        ; Specify X-Y endpoint position to create a 90 degree angle line segment
PLC0,0          ; Specify X and Y data, local coordinates
PL1             ; Specify local coordinate system
PARCOP0,0,5,0   ; Specify incremental X-Y endpoint position and X-Y center
                ; position for full circle clockwise arc
PL0             ; Specify work coordinate system
PLIN0,0         ; Specify X-Y endpoint position to create a line segment back to 0,0
END             ; End definition of path prog1
PCOMP prog1     ; Compile path prog1
PRUN prog1      ; Execute path prog1
```

PLC

Define Path Local Coordinates

Type	Path Contouring	**Product**	**Rev**
Syntax	`<!>PLC<r>,<r>`	6K	5.0
Units	1st r = X coordinate units (scalable by the SCLD value)		
	2nd r = Y coordinate units (scalable by the SCLD value)		
Range	0.00000 – ±999,999,999		
Default	n/a		
Response	No response - Must be defining a path (DEF)		
See Also	PAB, PL, SCLD, PWC, SCALE		

The Define Path Local Coordinates (`PLC`) command is used to specify the Local X -Y coordinate data required for subsequent segment definition in the Local coordinate system. This command places the X -Y coordinate value of the Local coordinate system at the beginning of the next segment. (The first `<r>` is the X coordinate, the second `<r>` is the Y coordinate.) This command must be used before the `PL1` command is given.

> **UNITS OF MEASURE** and **SCALING**: refer to page 16 or to the `SCLD` description.

Example: Refer to Define Path Local Mode (`PL`) command example.

PLCP

Compiled PLC Program

Type	PLC Scan Program	**Product**	**Rev**
Syntax	`<!>PLCPi`	6K	5.0
Units	i = number of PLC program		
Range	1-99		
Default	n/a		
Response	n/a		
See Also	DEF, ELSE, EXE, IF, L, LN, MEMORY, NIF, OUT, PCOMP, PEXE, PRUN, PUCOMP, SCANP, TSCAN, VARI, VARB		

`PLCP` is not a command; it is used to identify a PLCP program to be <u>defined</u> (e.g., `DEF PLCP2`), <u>compiled</u> (e.g., `PCOMP PLCP2`), and <u>executed</u> (e.g., `SCANP PLCP2` or `PRUN PLCP2`). Up to 99 PLCP programs may be defined, identified as `PLCP1`, `PLCP2`, `PLCP3`, and so on. The purpose of PLCP programs is to facilitate fast I/O scanning.

The process of creating and executing a PLCP program is:

1. Define the PLCP program (`DEF PLCPi` statement, followed by commands from the list below, followed by `END`). Only these commands are allowed in a PLCP program:

    - `IF`, `ELSE`, and `NIF` (conditional branching) — see note below for limitations
    - `L` and `LN` (loops)
    - `OUT` (turn on a digital output)
    - `EXE` (execute a program in a specific task — e.g., `2%EXE MOVE`)
    - `PEXE` (execute a program in a specific task — e.g., `3%PEXE PLCP4`)
    - `VARI` (integer variables).
    - `VARB` (binary variables). Bitwise operations are limited to Boolean And (`&`), Boolean Inclusive Or (`|`), and Boolean Exclusive Or (`^`).

2. Compile the PLCP program (`PCOMP PLCPi`). A compiled program runs much faster than a standard program.

3. Execute the PLCP program (`SCANP PLCPi`). When the PLCP program is launched with the `SCANP` command, it is executed in the "PLC Scan Mode". The advantage of the PLC Scan Mode is that the PLCP program is executed within a dedicated 0.5 ms time slot during every 2 ms system update period. This gives the PLCP program faster throughput for monitoring and manipulating I/O. *For more information on how the PLCP program is executed with* `SCANP`, *refer to the* `SCANP` *command description.*

    An alternative execution method is to use the `PRUN` command (`PRUN PLCPi`). This method is similar to the `SCANP PLCPi` method, but will only run through the PLCP program once.

Memory Requirements: Most commands allowed in a PLCP program consume one segment of compiled memory after the program is compiled with `PCOMP`; the exceptions are `VARI` and `VARB` (each consume 2 segments) and `IF` statements. Each `IF` conditional evaluation compounded with either an `AND` or an `OR` operator consumes an additional segment (e.g., `IF(IN.1=b1 AND 1AS.1=b0)` consumes three segments of compiled memory). The number of compounds is limited only by the memory available.

Conditional Expressions:

- Order of Evaluation. Because only one level of parenthesis is allowed, the order of evaluation of `IF` conditionals is from left to right. Refer to the flowchart for the evaluation logic.

- Conditional expressions in a PLC program use the non-scaled integer ("raw") operand values. Examples of the "raw" operand values are:
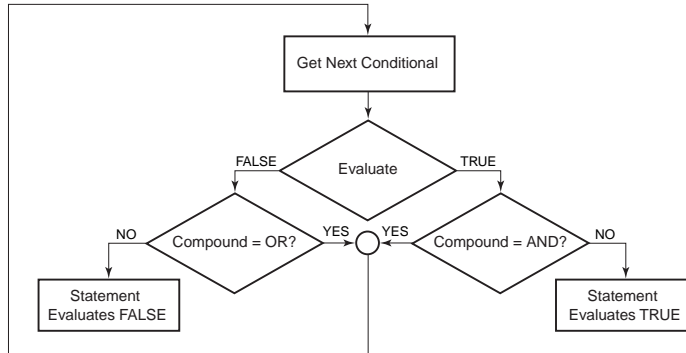
  - The `PE` operator reports encoder counts not scaled by `SCLD` and not scaled by `ERES`.
  - The `ANI` operator reports ADC counts from an analog input, not scaled by `SCLD`. Assuming the default `ANIRNG4` setting (+/-10V voltage range), 205 ADC counts = 1 volt.
  - The `DAC` operator reports DAC counts (commanded position) not scaled by `SCLD`.

  The only operands that are not allowed are: `SIN`, `COS`, `TAN`, `ATAN`, `VCVT`, `SQRT`, `VAR`, `TW`, `READ`, `DREAD`, `DREADF`, `DAT`, `DPTR`, and `PI`.

Programming Example: Refer to the detailed, illustrated example in the `SCANP` command description.

PLIN   Move in a Line

		Product	Rev
Type	`Path Contouring`	6K	5.0
Syntax	`<!><@>PLIN<r>,<r>`		
Units	`1st r = X endpoint coordinate (scalable by the SCLD value)`		
	`2nd r = Y endpoint coordinate (scalable by the SCLD value)`		
Range	`0.00000 – ±999,999,999`		
Default	`n/a`		
Response	`No response – Must be defining a path (DEF)`		
See Also	`PAB, PL, PLC, SCLD, PWC, SCALE`		

The Define Line Segment (`PLIN`) command is used to specify a line segment. The placement, length, and orientation of the line are completely specified by the endpoint of the line segment and the endpoint of the previous segment (current position). Segment endpoint position specifications may be either absolute (`PAB1`) with respect to the user defined coordinate system, or incremental (`PABØ`), relative to the start of each individual segment.

When the `PLIN` command is received, the first value is taken as the X endpoint coordinate and the second value is taken as the Y endpoint coordinate.

UNITS OF MEASURE and **SCALING**: refer to page 16 or to the `SCLD` description.

Example: Refer to Define Path Local Mode (`PL`) command example.

PLN   Loop End, Compiled Motion

Type	Compiled Motion	**Product**	**Rev**
Syntax	<@>PLN<b><b><b><b><b><b><b><b>	6K	5.0
Units	n/a		
Range	b = 1 (end loop), 0 or X (don't end loop)		
Default	n/a		
Response	No response; instead ends loop for compiled motion		
See Also	GOBUF, PCOMP, PLOOP, PRUN, PUCOMP		

The Loop End, Compiled Motion (PLN) command specifies the end of an axis-specific compiled motion profile loop, as initiated with the PLOOP command.

Programming Example: see PLOOP.

PLOOP   Loop Start, Compiled Motion

Type	Compiled Motion	**Product**	**Rev**
Syntax	<@>PLOOP<i>,<i>,<i>,<i>,<i>,<i>,<i>,<i>	6K	5.0
Units	i = designated number of loops for specified axis		
Range	0-2,147,483,647 (2^{31}-1)		
	0 = infinite loop		
Default	n/a		
Response	No response; instead starts loop for compiled motion		
See Also	GOBUF, PCOMP, PLN, PRUN, PUCOMP		

The PLOOP command specifies the beginning of an axis-specific profile loop. All subsequent segments defined before the PLN command are included within that loop. The number in a given axis field specifies the number of loops to be executed for that axis. If that number is a zero or blank, then the loop will be executed infinitely. The PLOOP command can be nested up to four levels deep within a program.

When using compiled loops (PLOOP and PLN), the last segment within the loop must end at zero velocity or there must be a final GOBUF segment placed outside (after) the loop. Otherwise an error will result when the profile is compiled. The error is "ERROR: MOTION ENDS IN NON-ZERO VELOCITY-AXIS n".

The PLOOP command will consume one segment of compiled space.

Example:
```
DEF prog1        ; Begin definition of prog1
V1               ; Set velocity to 1 unit/sec
D1000            ; Set distance to 1000 units
GOBUF1           ; Segment of motion sent to buffer

PLOOP3           ; Start loop of the subsequent move profile

V10              ; Set velocity to 10 units/sec
D25000           ; Set distance to 25000 units
GOBUF1           ; First segment within loop sent to buffer

V2               ; Set velocity to 2 units/sec
D1000            ; Set distance to 1000 units
GOBUF1           ; Second segment of motion within loop sent to buffer

V1               ; Set velocity to 1 unit/sec
D25000           ; Set distance to 25000 units
GOBUF1           ; Third segment within loop sent to buffer

PLN1             ; Close loop

V.5              ; Set velocity to 0.5 units/sec
D100             ; Set distance to 100 units
GOBUF1           ; Segment of motion sent to buffer (outside loop)

END              ; End definition of prog1

PCOMP prog1      ; Compile prog1
PRUN prog1       ; Execute prog1
```

[PMAS]   Current Master Cycle Position

Type	Following and 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	FMCNEW, FMCP, FOLMAS, FOLMD, [FS], GOWHEN, [PCMS], SCALE, SCLMAS, TPMAS, TFS		

The PMAS operator is used to assign the master position register value to a variable, or to make a comparison against another value. This value may be used for subsequent decision making, or for recording the cycle position corresponding to some other event.

PMAS is unique among position assignment variables, because its value rolls over to zero each time the entire master cycle length (FMCLEN value) has been traveled. If it is desired to WAIT or GOWHEN on a master cycle position of the next master cycle, one master cycle length (value of FMCLEN) should be added to the master cycle position specified in the argument. This allows commands that sequence follower events through a master cycle to be placed in a loop. The WAIT or GOWHEN command at the top of the loop could execute, even though the actual master travel had not finished the previous cycle. This is done to allow a PMAS value which is equal to the master cycle length to be specified and reliably detected. When using PMAS with IF, UNTIL, or WHILE arguments, the instantaneous PMAS value is used. Be careful to avoid specifying PMAS values that are nearly equal to the master cycle length (FMCLEN), because rollover may occur before a PMAS sample is read.

The master must be assigned first (FOLMAS **command) before this command will be useful.**

If scaling is enabled (SCALE1), the PMAS value is scaled by the master scaling factor (SCLMAS). If scaling is disabled (SCALEØ), the PMAS value is in counts.

Syntax: VARn=aPMAS where "n" is the variable number and "a" is the axis number, or PMAS can be used in an expression such as IF(2PMAS>2345Ø). The PMAS command must be used with an axis specifier, or it will default to axis 1 (e.g., VAR1=1PMAS, IF(2PMAS>5ØØ), etc.).

Example: (refer also to FOLEN example #2)
```
IF(2PMAS>4.3)    ; If the master for axis 2 has traveled more than 4.3
                 ; master user units then do the IF statement
OUT.2=b1         ; Set onboard output #2 to 1
NIF              ; End of IF statement
VAR14=1PMAS      ; Set VAR14 to axis 1's master cycle position
```

[PME]   Position of Master Encoder

Type	Assignment or Comparison	**Product**	**Rev**
Syntax	See below	6K	5.0
Units	Master Encoder counts		
Range	n/a		
Default	n/a		
Response	n/a		
See Also	MEPOL, MESND, [PCME], [PE], PMECLR, PMESET, TPCME, TPME		

Use the PME operator to assign the current master encoder position to a variable, or to make a comparison against another value. The master encoder is connected to the connector labeled "Master Encoder". If you issue a PMESET command, the encoder position value will be offset by the PMESET command value. The PME value is always in encoder counts, it is never scaled.

Syntax: VARn=PME where "n" is the variable number, or PME can be used in an expression such as IF(PME>16ØØØ).

Example:
```
VAR1=PME         ; Master encoder position is assigned to variable 1
IF(PME<4000)     ; If the master encoder count is less than 4000,
                 ; do the IF statement
VAR2=PME+4000    ; Master encoder position plus 4000 is assigned to variable 2
NIF              ; End IF statement
```

PMECLR   Clear Master Encoder Absolute Position Reference

Type	Motion	**Product**	**Rev**
Syntax	<!>PMECLR<r>	6K	5.0
Units	r = master encoder counts (not scalable)		
Range	±999,999,999.99999		
Default	n/a		
Response	n/a		
See Also	MEPOL, MESND, [PCME], [PME], PMESET, PSET, TPCME, TPME		

Use the PMECLR command to remove any offset on the master encoder position reports (offset to master encoder position is established with the PMESET command).

Example:
```
TPME             ; Report master encoder position. For the sake of this example,
                 ; let's say the response is *TPME10000 (master encoder is at
                 ; absolute position 10,000).
PMESET20000      ; Change relative position of master encoder with offset
TPME             ; Report new master encoder position with offset. New position
                 ; response should now be: *TPME20000 (what was considered
                 ; position 10,000 is now considered position 20,000).
PMECLR           ; Clear any offset applied to the master encoder position
TPME             ; Report master encoder position with no offsets.
                 ; Response should be: *TPME10000.
```

PMESET   Establish Master Encoder Absolute Position Reference

Type	Motion	**Product**	**Rev**
Syntax	<!>PMESET<r>	6K	5.0
Units	r = master encoder counts (not scalable)		
Range	±999,999,999.99999		
Default	n/a		
Response	n/a		
See Also	MEPOL, MESND, [PCME], [PME], PMECLR, PSET, TPCME, TPME		

Use the PMESET command to offset the current absolute position of the master encoder (connected to the connector labeled "Master Encoder") to establish an *absolute position reference*. To remove the offset, issue the PMECLR command.

All `PMESET` values entered are in master encoder counts; this value is never scaled.

Example:
```
TPME                ; Report master encoder position. For the sake of this example,
                    ; let's say the response is *TPME10000 (master encoder is at
                    ; absolute position 10,000).
PMESET20000         ; Change relative position of master encoder with offset
TPME                ; Report new master encoder position with offset. New position
                    ; response should now be: *TPME20000 (what was considered
                    ; position 10,000 is now considered position 20,000).
PMECLR              ; Clear any offset applied to the master encoder position
TPME                ; Report master encoder position with no offsets.
                    ; Response should be: *TPME10000.
```

PORT — Designate Destination Communication ("COM") Port

Type	Communication Interface	**Product**	**Rev**
Syntax	`<!>PORT<i>`	6K	5.0
Units	i = port number		
Range	1 (COM1), 2 (COM2)		
	NOTE: "COM1" is the "RS-232" or "ETHERNET" connector.		
	"COM2" is the "RS-232/485" connector.		
Default	1		
Response	n/a		
See Also	], [, BOT, DRPCHK, E, EOL, EOT, ERRDEF, ERRLVL, ERROK, ERRBAD, LOCK, [READ], WRITE, XONXOFF		

The Designate Destination Port (`PORT`) command is used to determine which COM port is affected by the `DRPCHK`, `E`, `ECHO`, `BOT`, `EOL`, `EOT`, `ERROK`, `ERRBAD`, `ERRDEF`, `ERRLVL`, and `XONOFF` commands. It also specifies the port to which responses and prompts from stored programs should be sent.

The `PORT` command also selects the target port through which the `WRITE` and `READ` commands transmit ASCII text strings. The `DWRITE` command (as well as all other RP240 commands) will affect the RP240 regardless of the `PORT` command setting. If no RP240 is detected, the commands are sent to the COM2 port. `DWRITE` text strings are always terminated with a carriage return.

Example (The `PORT` command can be used to designate `EOT` parameters for both ports. Assume that port COM1 is being used to communicate to the controller.)
```
PORT1           ; Select COM1 for EOT setup
EOT45,49,10     ; EOT for COM1 is -1<lf>
TPE             ; Send "Transfer Position of Encoder" response to COM1
                ; using EOT 45,49,10
PORT2           ; Select COM2 for EOT setup
EOT45,50,10     ; EOT for COM2 is -2<lf>
TPC             ; Send "Transfer Commanded Position" response to COM2
                ; using EOT 45,50,10
```

Example (The `PORT` command specifies both port setups and response destinations in a stored program.)
```
DEF qwe         ; Begin definition of qwe
PORT1
EOT45,49,10     ; EOT for COM1 is -1<lf>
TPE             ; Send "Transfer Position of Encoder" response to COM1
                ; using EOT 45,49,10
PORT2
EOT45,50,10     ; EOT for COM2 is -2<lf>
TPC             ; Send "Transfer Commanded Position" response to COM2
                ; using EOT 45,50,10
END             ; End definition of qwe
```

POUT   Compiled Output

			Product	Rev
Type	`Path Contouring; Compiled Motion`		6K	5.0
Syntax	`<!><B>POUT<n><b><b><b><b> ...`			
Units	`n = axis identifier letter (for compiled motion only);`			
	`b = enable bit specific outputs (see page 6)`			
Range	`n = A-H for axes 1-8, respectively (for compiled motion only);`			
	`b = 0 (off), 1 (on), or X (don't change)`			
Default	`0`			
Response	`n/a`			
See Also	`GOBUF, OUT, OUTEN, OUTFNC, OUTLVL, PCOMP, PRUN, PUCOMP`			

Use the `POUT` command to control outputs during Contouring Motion or Compiled Motion. The syntax for the `POUT` command depends on whether you are using it for Contouring or Compiled Motion:

Contouring: `POUT`	**Compiled Motion**:	`POUTA` (apply output pattern to the profile for axis #1)
		`POUTB` (apply output pattern to the profile for axis #2)
		`POUTC` (apply output pattern to the profile for axis #3)
		`POUTD` (apply output pattern to the profile for axis #4)
		`POUTE` (apply output pattern to the profile for axis #5)
		`POUTF` (apply output pattern to the profile for axis #6)
		`POUTG` (apply output pattern to the profile for axis #7)
		`POUTH` (apply output pattern to the profile for axis #8)

You may use the `POUT` command to control any of the onboard outputs, as well as any outputs on external I/O bricks, <u>as long as they are left in the default function (`OUTFNCi-A`)</u>. Refer to page 6 to understand how to address the outputs (onboard and on optional expansion I/O bricks) available on your 6K product.

If you attempt to change the state of an output that is <u>not</u> defined as an `OUTFNCi-A` (general-purpose) output, the controller will respond with an error message ("`OUTPUT BIT USED AS OUTFNC`") and the `POUT` command will not be executed (but command processing will continue).

If you wish to set only one output value, instead of all outputs, use the bit select (.) operator, followed by the number of the specified output. <u>Contouring example</u>: `2POUT.12-1` turns on only output 12 on I/O brick 2. <u>Compiled Motion example</u>: `2POUTA.12-1` turns on only output 12 on I/O brick 2 for the axis 1 profile.

The `POUT` command consumes one segment of compiled memory.

The programmable outputs are sampled once per "system update" (2 ms).

Contouring ONLY:

> The `POUT` command specifies the programmable output bit pattern to be applied to the outputs at the beginning of the next segment and remain throughout that segment. The `POUT` command may be issued before any segment definition command, and will affect all subsequent segments until a new `POUT` command is issued. A `POUT` command will not take affect if there is no segment definition command following it. To change the programmable outputs at the end of a path, the standard output (`OUT`) command must be used after the path is executed. These segment-defined output patterns are stored as part of the compiled path definition.

CONTOURING EXAMPLE: Refer to the `PARCOM` command example.

COMPILED MOTION EXAMPLES: (see next page)

COMPILED MOTION EXAMPLES:

```
OUTFNC3-A        ; Default output function for onboard output 3
OUTFNC6-A        ; Default output function for onboard output 6
DEF P1           ; Define program P1
D1000,25000      ; Set distance to travel
GOBUF11          ; Motion segments for axes 1 and 2
POUTA.3-1        ; Turn on onboard output 3 when axis 1 travels to 1000 steps
D2000,50000      ; New distance commanded
GOBUF11          ; Motion segments for axes 1 and 2
POUTA.3-0        ; Turn off onboard output 3 when axis 1 travels 2000
                 ; additional steps
POUTB.6-1        ; Turn on onboard output 6 when axis 2 travels to 75000 steps
D1000,25000      ; New distance commanded
GOBUF11          ; Motion segment for axes 1 and 2
POUTB.6-0        ; Turn off onboard output 6 when axis 2 travels 25000
                 ; additional steps
END              ; End program definition

PCOMP P1         ; Compiled program P1
PRUN P1          ; Execute program P1
```

When executing a Compiled Following profile, the `POUTn` statement is always executed as programmed. Therefore, in order to make sure an output is on for a given motion segment no matter what direction the master is traveling, you should use two `POUTn` statements (see example below).

```
POUTA.3-0        ; Turn off onboard output 3 for axis 1 - master going backwards
POUTA.3-1        ; Turn on onboard output 3 for axis 1 - master going forwards
GOBUF1           ; Motion segments for axes 1
POUTA.3-1        ; Turn on onboard output 3 for axis 1 - master going backwards
POUTA.3-0        ; Turn off onboard output 3 for axis 1 - master going forwards
```

If you desire to "pulse" an output (turn on for a given amount of time), then use the `POUTn` command along with the `GOWHEN(T=n)` command. For example:

```
POUTA.1-1        ; Turn on onboard output 1
GOWHEN(T=120)    ; Wait for 120 milliseconds
POUTA.1-0        ; Turn off onboard output 1
```

PPRO   Path Proportional Axis

Type	Path Contouring	**Product**	**Rev**
Syntax	`<!>PPRO<r>`	6K	5.0
Units	r = ratio value		
Range	±0.001 – 1000.000		
Default	n/a		
Response	No response – Must be defining a path (DEF)		
See Also	PAXES		

The Path Proportional Axis (`PPRO`) command is used to specify the proportional axis to path travel ratio. The proportional axis will keep a position that is proportional to the distance traveled along the X-Y path as the path is executed. This allows the proportional axis to act as the Z axis in helical interpolation or to control the motion of any object which moves with distance and velocity proportional to the path.

The `PPRO` command should be given prior to any contour segments during a path definition. A negative value for the proportional axis ratio simply causes motion in the negative direction as path travel in the X-Y plane gets larger.

Example: (see contouring programming example in the `PRUN` command description)

PRTOL   Path Radius Tolerance

Type	Path Contouring	**Product**	**Rev**
Syntax	`<!>PRTOL<r>`	6K	5.0
Units	r = allowable radius error (scalable by the SCLD value)		
Range	±999,999,999.99999		
Default	1		
Response	No response - Must be defining a path (DEF)		
See Also	PARCM, PARCOM, PARCOP, PARCP, SCLD, SCALE		

The Path Radius Tolerance (PRTOL) command is used to specify the allowable radius error that is encountered when contouring.

The radius error is encountered in one of two ways. The first way is through use of the PARCM or PARCP commands. This error is the difference between the radius value specified in the PARCM or PARCP command and the minimum radius implied by the starting point and endpoint. If the radius provided in the command is smaller than the minimum radius implied by the distance from starting to endpoints and the error is within the radius tolerance then just enough is added to the radius to make a half circle.

A second way to encounter a radius tolerance error is with the PARCOM or PARCOP commands. This error is the difference between the radius implied by the start point and center point and the radius implied by the end point and center point. If the difference in the two radius values is within the radius tolerance specified, then the center point is moved such that an arc can be traveled through the start point and endpoint. The PRTOL command can be executed many times within a path definition allowing some arcs to be exactly known and others to be approximated.

If the radius error exceeds the PRTOL value, an error message is sent.

> **UNITS OF MEASURE** and **SCALING**: refer to page 16 or to the SCLD description.

Example:
```
PV5                ; Set path velocity to 5 units/sec
PA50               ; Set path acceleration to 50 units/sec/sec
PAD100             ; Set path deceleration to 100 units/sec/sec
DEF prog1          ; Begin definition of path named prog1
PAXES1,2           ; Set axes 1 and 2 as the X and Y contouring axes
PAB0               ; Set to incremental coordinates
PRTOL0.001         ; Allow 25 steps (0.001 x 25000) radius error
PARCM5,5,5         ; Specify incremental X-Y endpoint position and radius
                   ; arc <180 degree for quarter circle counter-clockwise arc
PARCP5,-5,-5       ; Specify incremental X-Y endpoint position and radius
                   ; arc >180 degree for three quarter circle clockwise arc
END                ; End definition of path prog1
PCOMP prog1        ; Compile path prog1
PRUN prog1         ; Execute path prog1
```

PRUN   **Run a Compiled Profile**

Type	Compiled Motion; Path Contouring; PLC Program	**Product**	**Rev**
Syntax	`<!>PRUN<t>`	6K	5.0
Units	t = text (name of path program)		
Range	text name of 6 characters or less		
Default	n/a		
Response	n/a		
See Also	COMEXC, DEF, END, PCOMP, PUCOMP, GOBUF, PLCP, PLOOP, PLN, SCANP		

Use the `PRUN` command to start execution of a previously compiled program (multi-axis contour, a `GOBUF` profile, or a `PLCP` program). All the required information about the program or path whose name is specified in the `PRUN` command has already been stored by the definition commands (`DEF` and `END`) and compiled by the `PCOMP` command.

<u>Executing compiled contouring and `GOBUF` profiles</u>: If any of the axes included in the specified path or profile are not ready, the path will not be executed. An axis is not ready if it is shutdown, moving, or in joystick mode. When execution of a pre-compiled program begins, all included axes become busy until motion has completed.

<u>Executing PLC programs</u>: When using `PRUN` to execute a `PLCP` program, the `PLCP` program is run only once (as opposed to invoking a continual scan loop when executing `PLCP` programs with the `SCANP` command).

`COMEXC1` mode must be enabled in order for command processing to continue once a motion invoking command has been initiated with `PRUN`. If you use the `PRUN` command within a program while in `COMEXC1` mode, it functions as a `GO` and returns control back to the original program after the embedded program's motion is started (control is returned to the first command immediately following the `PRUN` command). If in `COMEXCØ` mode, command processing will not continue until the motion invoking command has completed its movement.

CONTOURING EXAMPLE:
```
DEL prog1          ; Delete prog1
DEF prog1          ; Begin definition of path named prog1
PAXES1,2,3,4       ; Set axes 1,2,3,4 as the X, Y, Tangent, and
                   ; Proportional axes respectively
PPRO2.25           ; Proportional axis path ratio = 2.25
;  *********************************************
;  *  Add multiple path segment                *
;  *  definitions in this                       *
;  *  portion of the                            *
;  *  program                                   *
;  *********************************************
END                ; End definition of path prog1
PCOMP prog1        ; Compile path prog1
PRUN prog1         ; Execute path prog1
```


COMPILED MOTION EXAMPLE:
```
@D25000            ; Set distance parameter for all axes
DEL prog1          ; Delete prog1
DEF prog1          ; Define prog1
PLIN1000,1000      ; Line segment on axis 1 and 2
GOBUFxx11          ; Compiled motion on axis 3 and 4
END                ; End definition of prog1
PCOMP prog1        ; Compile prog1
PRUN prog1         ; Execute prog1
TPC                ; Check commanded position. A sample response would be:
                   ; "*TPM1000,1000,25000,25000"
```

PS   **Pause Program Execution**

Type	Program Flow Control		**Product**	**Rev**
Syntax	`<!>PS`		6K	5.0
Units	n/a			
Range	n/a			
Default	n/a			
Response	n/a			
See Also	C, COMEXR, COMEXS, K, S, [SS], TSS			

The Pause Program Execution (`PS`) command pauses execution of commands in the command buffer. If a `PS` command is executed, no commands after the `PS` will be executed until a `!C` command is received. However, additional commands may still be placed in the command buffer.

The `PS` command does not pause motion. In order for motion to be paused, the `S` and the `COMEXS` commands should be used.

Example:
```
PS                ; Stop execution of command buffer until !C command
MA0XXX            ; Incremental mode for axis 1
D10000            ; Set distance to 10000 units on axis 1
GO1000            ; Initiate motion on axis 1
D,20000           ; Set distance to 20000 units on axis 2
GO0100            ; Initiate motion on axis 2
; *****************************************************************
; * NOTE:                                                        *
; * No commands after the PS command will be executed until a !C *
; * command is received.                                         *
; *****************************************************************
```

PSET   **Establish Absolute Position**

Type	Motion		**Product**	**Rev**
Syntax	`<!><@>PSET<r>,<r>,<r>,<r>,<r>,<r>,<r>,<r>`		6K	5.0
Units	r = units (absolute position)			
Range	±999,999,999.99999			
Default	n/a			
Response	n/a			
See Also	CMDDIR, D, ENCPOL, [FB], GO, HOM, INFNC, MA, MC, [PANI], [PC], [PCC], [PCE], [PCMS], [PE], PESET, PMESET, SCALE, SCLD, SFB, TFB, TPANI, TPC, TPCC, TPCE, TPCMS, TPE			

Use the `PSET` command to offset the current absolute position to establish an *absolute position reference*. To remove the offset, issue the `PSET CLR` command. All `PSET` values entered are in steps, unless scaling is enabled (`SCALE1`), in which case (`PSET`) is multiplied by the distance scale factor (`SCLD`).

Steppers – without scaling: The `PSET` command will define the current commanded position to be the absolute position entered. To set an absolute encoder position, use the `PESET` command.

Servos – without scaling: The `PSET` command defines a new absolute position reference. If the drive is enabled (`DRIVE`), the current commanded position is used as the reference point. If the drive is disabled, the current feedback device position (selected with the `SFB` command) is used as the reference point.

SERVO AXES

The `PSET` offset value (per axis) is specific only to the feedback source (per axis) selected with the last `SFB` command.

If your application requires switching between feedback sources for the same axis, then you must select the feedback source with the appropriate `SFB` command and issue a `PSET` value specific to that feedback source. (Each feedback source can have a separate offset.)

NOTE: If you issue a `PSET` command, any previously captured positions (`INFNCi-H` or `LIMFNCi-H` function) will be offset by the `PSET` value.

If a software end-of-travel limit has been hit, the `PSET` command will not remove the error condition. The error condition is removed by commanding motion in the opposite direction.

Example:
```
PSET0,0,0,1000  ; Set absolute position on axes 1, 2, and 3 to zero,
                ; and axis 4 to 1000 units
```

[PSHF]   Net Position Shift

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	FOLEN, FOLMAS, FSHFC, FSHFD, SCALE, SCLD, TPSHF		

The `PSHF` operator is used to assign to a numeric variable the value of the net (absolute) follower axis position shift that has occurred since that last `FOLEN1` command. The position value will be the sum of all shifts performed on that axis, or axes, including decelerations due to limits, kill, or stop. The shift value is set to zero each time a new `FOLEN1` command or a `FOLMAS` command (with a value other than zero) is issued.

If scaling in enabled (`SCALE1`), the `PSHF` value is scaled by the distance scaling factor (`SCLD`). If scaling is not enabled, the value is in commanded counts.

Syntax: `VARn=aPSHF` where "n" is the variable number and "a" is the axis number, or `PSHF` can be used in an expression such as `IF(2PSHF>2345Ø)`. The `PSHF` command must be used with an axis specifier, or it will default to axis 1 (e.g., `VAR1=1PSHF`, `IF(2PSHF>5ØØ)`, etc.).

Example:
```
IF(2PSHF>4.3)    ; If axis 2 has shifted more than 4.3 user units in the
                 ; positive direction, then do the IF statement
OUT.2=b1         ; Turn on onboard output #2
NIF              ; End of IF statement
VAR14=3PSHF      ; Set VAR14 to follower axis 3's position shift
```

[PSLV]   Current Commanded Position of Follower Axis

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	FMCNEW, FMCP, SCLD, SCALE, TPSLV		

Use the `PSLV` operator to assign the follower axis commanded position register value to a variable, or to make a comparison against another value.

If scaling in enabled (`SCALE1`), the `PSLV` value is scaled by the distance scaling factor (`SCLD`). If scaling is not enabled, the value is in commanded counts.

Syntax: `VARn=aPSLV` where "n" is the variable number and "a" is the axis number, or `PSLV` can be used in an expression such as `IF(2PSLV>2345Ø)`. The `PSLV` command must be used with an axis specifier, or it will default to axis 1 (e.g., `VAR1=1PSLV`, `IF(2PSLV>5ØØ)`, etc.).

Example:
```
IF(2PSLV>4.3)    ; If axis 2 has traveled more than 4.3 user units then do
                 ; the IF statement
OUT.2=b1         ; Turn on onboard output #2
NIF              ; End of IF statement
VAR14=3PSLV      ; Set VAR14 to follower axis #3's position
```

PTAN   **Path Tangent Axis Resolution**

Type	Path Contouring	**Product**	**Rev**
Syntax	`<!>PTAN<i>`	6K	5.0
Units	i = counts (commanded counts for stepper axes, encoder or analog input counts for servo axes)		
Range	±1 – 999,999,999		
Default	4000		
Response	No response - Must be defining a path (DEF)		
See Also	PAXES		

The Path Tangent Axis Resolution (PTAN) command is used to specify the Tangent axis resolution. The Tangent axis will keep an angular position which changes linearly with the direction of travel implied by X and Y. This allows the Tangent axis to control an object which must stay tangent (or normal) to the direction of travel.

The Tangent axis resolution is the number of counts (motor steps for steppers; encoder or analog input counts) in 360 degrees of arc. The Tangent axis resolution does not necessarily equal axis resolution (DRES for steppers; ERES or analog input counts/volt for servos), but if the motor directly drove the rotating piece, then these numbers would be the same.

The PTAN command should be given prior to any contour segments during a path definition. A negative value for the Tangent axis resolution causes rotation in the negative direction as the angle in the X-Y plane gets larger.

Example:
```
PV5               ; Set path velocity to 5 units/sec
PA50              ; Set path acceleration to 50 units/sec/sec
PAD100            ; Set path deceleration to 100 units/sec/sec
DEF prog1         ; Begin definition of path named prog1
PAXES1,2,3        ; Set axes 1 and 2 as the X and Y contouring axes,
                  ; 3 as the tangent axis
PTAN25000         ; Specify Tangent axis resolution
PAB0              ; Set to incremental coordinates
POUT1001          ; Output pattern during first arc (onboard outputs)
PARCM5,5,5        ; Specify incremental X-Y endpoint position and radius
                  ; arc <180 degree for quarter circle counter-clockwise arc
POUT1100          ; Output pattern during second arc (onboard outputs)
PARCP5,-5,-5      ; Specify incremental X-Y endpoint position and radius
                  ; arc >180 degree for three quarter circle clockwise arc
END               ; End definition of path prog1
PCOMP prog1       ; Compile path prog1
PRUN prog1        ; Execute path prog1
OUT0000           ; Turn off the first four onboard outputs
```

PUCOMP   Un-Compile a Compiled Profile (includes Path Uncompile)

		Product	Rev
Type	Compiled Motion; Path Contouring; PLC Program	6K	5.0
Syntax	<!>PUCOMP<t>		
Units	t = text (name of path)		
Range	Text name of 6 characters or less		
Default	n/a		
Response	n/a		
See Also	DEF, END, GOBUF, MEMORY, PCOMP, PLCP, PRUN, SCANP, TDIR, TMEM, TSEG, GOBUF, PLOOP, PLN		

The Un-Compile (PUCOMP) command is used to delete a previously compiled (PCOMP) program from the compiled memory. The PUCOMP command does not delete the program from program memory.

Example:
```
PUCOMP prog1     ; Delete compiled motion segments for prog1
DEL prog1        ; Delete prog1
DEF prog1        ; Begin definition of path named prog1
PAXES1,2,3,4     ; Set axes 1,2,3,4 as the X, Y, Tangent, and
                 ; Proportional axes respectively
;   *******************************************
;   *  Add multiple path segment              *
;   *  definitions in this                    *
;   *  portion of the                         *
;   *  program                                *
;   *******************************************
END              ; End definition of path prog1
PCOMP prog1      ; Compile path prog1
PRUN prog1       ; Execute path prog1
```

PULSE   Pulse Width

		Product	Rev
Type	Controller Configuration	6K	5.0
Syntax	<!><@><a>PULSE<r>,<r>,<r>,<r>,<r>,<r>,<r>,<r>		
Units	r = microseconds (µs)	(applicable to stepper axes only)	
Range	0.3, 0.5, 1.0, 2.0, 4.0, 8.0, or 16.0		
Default	0.3		
Response	PULSE: *PULSE0.3,0.3,0.3,0.3,0.3,0.3,0.3,0.3 1PULSE: *1PULSE0.3		
See Also	AXSDEF, DRES, V		

The Pulse Width (PULSE) command sets the step output pulse width. The pulse width is described as the time the pulse is active, or *on*. The value for the pulse width command is specified in microseconds.

When the pulse width is changed from the default value of 0.3 µs, the maximum velocity range is reduced. The amount of reduction is directly proportional to the change in pulse width (see table below).

Pulse Width (PULSE) Setting	Actual Pulse Width	Maximum Velocity
DEFAULT → 0.3 µs	0.244 µs	2.048 MHz
0.5 µs	0.484 µs	1.024 MHz
1.0 µs	0.976 µs	512 KHz
2.0 µs	1.953 µs	256 KHz
4.0 µs	3.906 µs	128 KHz
8.0 µs	7.812 µs	64 KHz
16.0 µs	15.624 µs	32 KHz

PV   Path Velocity

		Product	**Rev**
Type	`Path Contouring or Motion (Linear Interpolated)`	6K	5.0
Syntax	`<!>PV<r>`		
Units	`r = units/sec (scalable with SCLD)`		
Range	`Stepper Axes: 0.00000-2,048,000 (max. depends on SCLD & PULSE)`		
	`Servo Axes:   0.00000-6,500,000 (max. depends on SCLD)`		
Default	`1.0000`		
Response	`PV:    *PV1.0000`		
See Also	`GOL, SCLD, SCALE`		

The Path Velocity (`PV`) command specifies the path velocity to be used in linearly interpolated moves (`GOL`), and in all contouring moves. In linearly interpolated moves, a path may involve one to four axes, each with its own distance of travel. In contouring paths, only the X and Y axis are included in the calculation of the path.

For both types of moves, the path velocity refers to the velocity of the load as motion proceeds along the path. For linearly interpolated moves, the velocity of each individual axis is dependent on the distance it contributes to the total path traveled by the load. In contouring paths, the velocity of each individual axis is dependent on the direction of travel in the X- Y plane. ***NOTE***: *The* `PV` *value can be altered between path segments, but not within a path segment.*

> **UNITS OF MEASURE** and **SCALING**: refer to page 16.

Example: Refer to Define Path Local Mode (`PL`) command example.

PWC   Path Work Coordinates

		Product	**Rev**
Type	`Path Contouring`	6K	5.0
Syntax	`<!>PWC<r>,<r>`		
Units	`1`st` r = X coordinate units (scalable by the SCLD value)`		
	`2`nd` r = Y coordinate units (scalable by the SCLD value)`		
Range	`±999,999,999.99999`		
Default	`0,0`		
Response	`No response - Must be defining a path (DEF)`		
See Also	`PAB, PL, PLC, SCLD, SCALE`		

The Path Work Coordinates (`PWC`) command is used to specify the Work X -Y coordinate data required for subsequent segment definition in the Work coordinate system. This command places the X -Y coordinate value of the Work coordinate system at the beginning of the next segment. (The first `<r>` is the X coordinate, the second `<r>` is the Y coordinate.)

This command may be used before the `PLØ` command is given for the purpose of shifting the Work coordinate system. If the `PWC` command is not given before a `PLØ` command, but was previously set, the original work coordinate system is used for the subsequent segments.

> **UNITS OF MEASURE** and **SCALING**: refer to page 16 or to the `SCLD` description.

Example: Refer to Define Path Local Mode (`PL`) command example.

RADIAN   Radian Enable

		Product	**Rev**
Type	Operators(Trigonometric)	6K	5.0
Syntax	<!>RADIAN<b>		
Units	n/a		
Range	b = 0 (Disable),1 (Enable) or X (don't care)		
Default	0		
Response	RADIAN: *RADIAN0		
See Also	[ATAN],[COS],[PI],[SIN],[TAN],VAR		

This operator is used to switch between radians and degrees. The command RADIAN1 specifies units in radians for SIN, COS, TAN, and ATAN. The command RADIANØ specifies units in degrees for SIN, COS, TAN, and ATAN.

If a value is given in radians and a conversion is needed to degrees, use the formula: $360° = 2\pi$ radians.

Example:
```
RADIAN1            ; Set trigonometric functions to radian mode
```

RE   Registration Enable

		Product	**Rev**
Type	Registration	6K	5.0
Syntax	<!><@><a>RE<b><b><b><b><b><b><b>		
Units	b = 0 (disable), 1 (enable), or X (don't care)		
Range	n/a		
Default	0		
Response	RE: *RE0000_0000		
	1RE: *1RE0		
See Also	[AS], COMEXC, ENCCNT, [ER], INFNC, [PCC], [PCE], [PCMS], REG, REGLOD, REGSS, TAS, TER, TPCC, TPCE, TPCMS, TRGLOT, [TRIG], TTRIG		

The Registration Enable (RE) command enables the registration function for the specified axes.

When a registration input (a trigger input assigned the "Trigger Interrupt" function) is activated, the motion profile currently being executed is replaced by a *registration profile* with its own distance (REG), acceleration (A & AA), deceleration (AD & ADA), and velocity (V) values. The registration move may interrupt any preset, continuous, or registration move in progress.

The registration move does not alter the rest of the program being executed when registration occurs, nor does it affect commands being executed in the background if the controller is operating in the continuous command execution mode (COMEXC1).

Registration moves will not be executed while the motor is not performing a move, while in the joystick mode (JOY1), or while decelerating due to a stop, kill, soft limit, or hard limit.

How to Set up a Registration Move

1. Configure one of the trigger inputs (TRG-nA or TRG-nB per axis, or TRG-M) to function as a trigger interrupt input; this is done with the INFNCi-H command, where i is the input bit number representing the targeted trigger input.

2. Specify the distance of the registration move with the REG command. For servo axes, the distance refers to the encoder position (not functional with ANI feedback). For stepper axes, the distance refers to commanded position.

3. Enable the registration function with the RE command. Registration is performed only on the axis or axes with the registration function enabled, and with a non-zero distance specified in the respective axis-designation field of the REG command; the other axes will not be affected. Each trigger has a distinct move defined for its dedicated axis.

   NOTE: The registration move is executed using the A, AA, AD, ADA, and V values that were in effect when the REG command was entered.

Registration Move Accuracy (see also Registration Move Status below)

The accuracy of the registration move distance specified with the `REG` command is ±1 count (servo axes: encoder count; stepper axes: commanded count if `ENCCNT0` or encoder count if `ENCCNT1`).

RULE OF THUMB: To prevent position overshoot, make sure the `REG` distance is greater than 4 ms multiplied by the incoming velocity.

The lapse between activating the registration input and commencing the registration move (this does not affect the move accuracy) is less than one position sample period (2 ms).

The `REG` distance will be scaled by the distance scale factor (`SCLD` value) if scaling is enabled (`SCALE1`). See page 16 for details on scaling.

Preventing Unwanted Registration Moves (methods)

- Registration Input Debounce: Registration Input Debounce: By default, the registration inputs are debounced for 24 ms before another input on the same trigger is recognized. (The debounce time is the time required between a trigger's initial active transition and its secondary active transition.) Therefore, the maximum rate that a registration input can initiate registration moves is 500 times per second. If your application requires a shorter debounce time, you can change it with the `TRGLOT` command.

- Registration Single-Shot: The `REGSS` command allows you to program the 6K controller to ignore any registration commands after the first registration move has been initiated. Refer to the `REGSS` command description for further details and an application example.

- Registration Lockout Distance: The `REGLOD` command specifies what distance an axis must travel before any trigger assigned as a registration input will be recognized. Refer to the `REGLOD` command description for further details and an application example.

Registration Move Status & Error Handling

Axis Status — Bit #28: This status bit is set when a registration move has been initiated by any registration input (trigger). This status bit is cleared with the next `GO` command.

> `AS.28`.......Assignment & comparison operator — use in a conditional expression.
> `TASF`.........Full text description of each status bit. (see "`Reg Move Commanded`" line item)
> `TAS`...........Binary report of each status bit (bits 1-32 from left to right). See bit #28.

Axis Status — Bit #30: If, when the registration input is activated, the registration move profile cannot be performed with the specified motion parameters, the 6K controller will kill the move in progress and set axis status bit #30. This status bit is cleared with the next `GO` command.

> `AS.30`.......Assignment & comparison operator — use in a conditional expression.
> `TASF`.........Full text description of each status bit. (see "`Preset Move Overshot`" line item)
> `TAS`...........Binary report of each status bit (bits 1-32 from left to right). See bit #30.

Error Status — Bit #10: This status bit may be set if axis status bit #30 is set. The error status is monitored and reported only if you enable error-checking bit #10 with the `ERROR` command (e.g., `ERROR.10-1`). NOTE: When the error occurs, the controller will branch to the error program (assigned with the `ERRORP` command). This status bit is cleared with the next `GO` command.

> `ER.10`.......Assignment & comparison operator — use in a conditional expression.
> `TERF`.........Full text description of each status bit. (see "`Preset Move Overshot`" line item)
> `TER`...........Binary report of each status bit (bits 1-32 from left to right). See bit #10.

Trigger Status — Bits #1-17: Trigger status bits are set when a registration move has been initiated by trigger inputs A or B for each axis, or with the TRIG-M (master trigger) input. This also indicates that the positions of all axes has been captured. As soon as the captured information is transferred or assigned/compared, the respective trigger status bit is cleared (set to ∅).

> `TRIG`.........Assignment & comparison operator — use in a conditional expression.
> `TTRIG`.......Binary report of each status bit (bits 1-17 from left to right). From left to right the bits represent trigger A and B for axes 1-8, the 17th bit is master trigger M (the "`MASTER TRIG`" input terminal) — see page 7.

Example:

In this example (using axis 1), two-tiered registration is achieved. While axis 1 is executing it's 50,000-unit move, trigger input 1A is activated and executes registration move A to slow the load's movement. An open container of volatile liquid is then placed on the conveyor belt. After picking up the liquid and while registration move A is still in progress, trigger input 1B is activated and executes registration move B to slow the load to gentle stop.

```
DEL REGI1        ; Delete program (assume program already resides in memory)
DEF REGI1        ; Begin program definition
INFNC1-H         ; Define trigger input 1A (axis 1) as a trigger interrupt input
INFNC2-H         ; Define trigger input 1B (axis 1) as a trigger interrupt input
A20              ; Set acceleration on axis 1 to 20 units/sec/sec
AD40             ; Set deceleration on axis 1 to 40 units/sec/sec
V1               ; Set velocity on axis 1 to 1 unit/sec
1REGA4000        ; Set trigger 1A's registration distance on axis 1 to 4000 units
                 ; (registration A move will use the A, AD, & V values above)
A5               ; Set acceleration on axis 1 to 5 units/sec/sec
AD2              ; Set deceleration on axis 1 to 2 units/sec/sec
V.5              ; Set velocity on axis 1 to 0.5 units/sec
1REGB13000       ; Set trigger 1B's registration distance on axis 1 to 13,000 units
                 ; (registration B move will use the A, AD, & V values above)
RE1              ; Enable registration on axis 1 only
A50              ; Set acceleration to 50 units/sec/sec on axis 1
AD50             ; Set deceleration to 50 units/sec/sec on axis 1
V10              ; Set velocity to 10 unit/sec on axis 1
D50000           ; Set distance to 50000 units on axis 1
GO1              ; Initiate motion on axis 1
END              ; End program definition
```

Registration Profile:
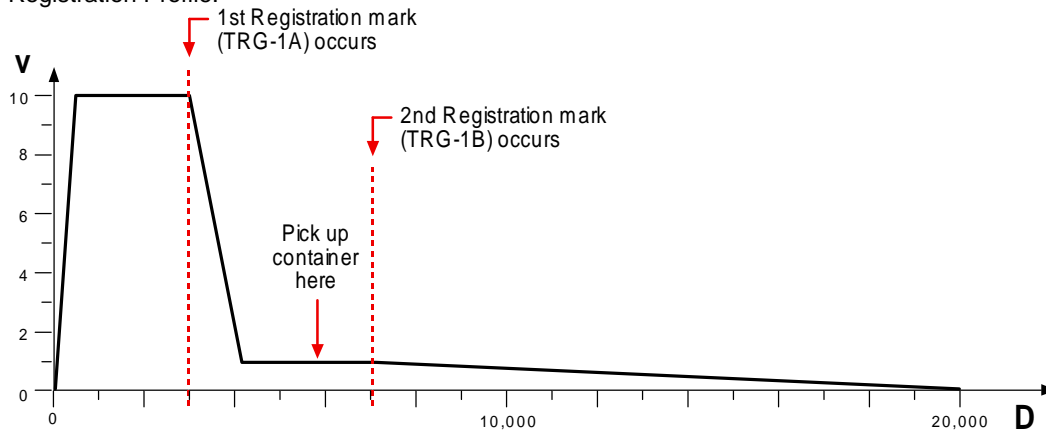
[READ]   Read a Value

Type	Communication Interface or Assignment
Syntax	`... READi ... (See below)`
Units	`i = string variable number`
Range	`1-50`
Default	`n/a`
Response	`n/a`
See Also	`', PORT, [ SS ], TSS, VAR, VARS, WRITE`

Product **Rev**
6K   5.0

The Read a Value (`READ`) command provides the user with an efficient way of storing numeric data read from the input buffer into a variable. The `READ` command can be used as part of a numeric variable assignment statement (e.g., `VAR1=READ1`) or in another command (`A1Ø,(READ1),12,1`). However, the `READ` command cannot be used in an expression such as `VAR5=1+READ1` or `IF(READ1=1)`.

Syntax: `VARx=READi` where `x` is the variable number and `i` is the string variable to be sent out to prompt the user for the numeric information.

Syntax: `Command(READi)` where `Command` is any command that has a separate field (e.g., `A`, `AD`, `V`, `D`, etc.), and `i` is the string variable number.

The number attached to the end of the `READ` command corresponds to the string variable to be sent out the Ethernet port or the RS-232 or RS-485port, at the time this command is executed. The 6K Series controller will then wait for numeric data to be sent to its input buffer. **The numeric data must be preceded with an immediate command identifier and a single quote (`!'`).** The information read in can be either integer, or real, and must be terminated by a command delimiter (`:`, `<cr>`, `<lf>`).

<u>Rule of Thumb for command value substitutions</u>: If the command syntax shows that the command field requires a real number (denoted by `<r>`) or and integer value (denoted by `<i>`), you can use the `READ` substitution (e.g., `V2,(READ)`).

Example:
```
VARS1="Enter the count >" ; Place message in string variable #1
VAR2=READ1               ; Prompt with string variable #1, and read data
                         ; into variable #2
;
; The controller will send this message (string variable #1) to the screen:
;               "Enter the count >"
; The user must enter the numeric data preceded by the characters !'.
; For example, !'82.5 assigns the value 82.5 to numeric variable 2
```

REG   Registration Distance

Type	Registration	**Product**	**Rev**
Syntax	<!><@>aREGc<r>	6K	5.0

Units	a = axis #
	c = letter of trigger input
	r = distance units (scalable by the SCLD value)
	servo axes: always encoder counts (ANI input not allowed)
	stepper axes: commanded counts
Range	a = 1-8 (depending on product)
	c = A or B
	r = 0.00000 to 419,430,000.00000 (positive direction only)
Default	0 (do not make a registration move)
Response	1REGA: *1REGA0
See Also	[AS], ENCCNT, [ER], [PCC], [PCE], [PCMS], RE, REGLOD, REGSS, SCALE, SCLD, [SS], TAS, TER, TPCE, TRGLOT, [TRIG], TTRIG

The Registration Distance (REG) command specifies the distance the corresponding axis will travel after receiving a registration input (trigger A or B). Example: 1REGA4000 sets up a 4000-count registration move on axis 1 to be initiated when trigger input 1A is activated.

Servo Axes: REG value always represents encoder counts (registration cannot be used with analog input feedback).

Stepper Axes: REG value represents commanded counts.

Trigger Input (Axis 1-4 "TRIGGERS/OUTPUTS" connector) *		Dedicated Axis	REG Syntax	Trigger Input (Axis 5-8 "TRIGGERS/OUTPUTS" connector) *		Dedicated Axis	REG Syntax
Pin 23,	Trigger 1A	1	1REGA	Pin 23,	Trigger 5A	5	5REGA
Pin 21,	Trigger 1B	1	1REGB	Pin 21,	Trigger 5B	5	5REGB
Pin 19,	Trigger 2A	2	2REGA	Pin 19,	Trigger 6A	6	6REGA
Pin 17,	Trigger 2B	2	2REGB	Pin 17,	Trigger 6B	6	6REGB
Pin 15,	Trigger 3A	3	3REGA	Pin 15,	Trigger 7A	7	7REGA
Pin 13,	Trigger 3B	3	3REGB	Pin 13,	Trigger 7B	7	7REGB
Pin 11,	Trigger 4A	4	4REGA	Pin 11,	Trigger 8A	8	8REGA
Pin 9,	Trigger 4B	4	4REGB	Pin 9,	Trigger 8B	8	8REGB

* The number of trigger inputs available varies by product (refer to your product's *Installation Guide*).

The registration move is executed using the A, AA, AD, ADA, and V values that were in effect when the REG command was entered.

RULE OF THUMB: To prevent position overshoot, make sure the REG distance is greater than 4 ms multiplied by the incoming velocity.

The registration distance remains set until you change it with a subsequent REG command. Registration distances outside the valid range are flagged as an error, returning the message *INVALID DATA-FIELD x, where x is the field number.

UNITS OF MEASURE and **SCALING**: refer to page 16.

For additional details on Registration (including programming examples), refer to the RE command description and to the Registration section in the *Programmer's Guide*.

REGLOD   Registration Lock-Out Distance

		Product	Rev
Type	Registration		
Syntax	`<!><@>REGLOD<r>,<r>,<r>,<r>,<r>,<r>,<r>,<r>`	6K	5.0
Units	r = distance units (scalable by SCLD)		
Range	0.00000 to +999,999,999.99999		
Default	0		
Response	REGLOD: *REGLOD0,0,0,0,0,0,0,0		
	1REGLOD: *1REGLOD0		
See Also	INFNC, RE, REG, REGSS, SCLD, TRGLOT		

The `REGLOD` command specifies the distance an axis must travel before its registration input will be recognized. If scaling is enabled (`SCALE1`), the lock-out distance is scaled by the `SCLD` value.

Stepper axes: The lock-out distances are measured incrementally from the start of motion to the <u>commanded position</u>.

Servo axes: The lock-out distances are measured incrementally from the start of motion to the <u>actual position</u> (as measured by the position feedback device), not the commanded position.

Example (single axis, stepper)**:**
A print wheel uses registration to initiate each print cycle. From the beginning of motion, the controller should ignore all registration marks before traveling 2000 steps. This is to ensure that the unit is up to speed and that the registration mark is a valid one.

```
DEL REGI3         ; Delete program (in case program already resides in memory)
DEF REGI3         ; Begin program definition
INFNC1-H          ; Trigger capture mode for trigger 1A on axis 1
RE1               ; Enable registration
V2                ; Set registration move to a velocity of 2 revs/sec
1REGA2500         ; and a distance of 2500 steps
REGLOD2000        ; Set registration lockout distance to 2000 steps
MC1               ; Start a mode continuous move
V1                ; move at a velocity of 1 rps
GO1               ; Initiate motion
END               ; End program definition
```

Registration Profile:

1st Registration mark occurs after 1500 steps, but the registration move does not occur because the lockout distance is set to 2000 steps.

2nd Registration mark occurs after 3000 steps.
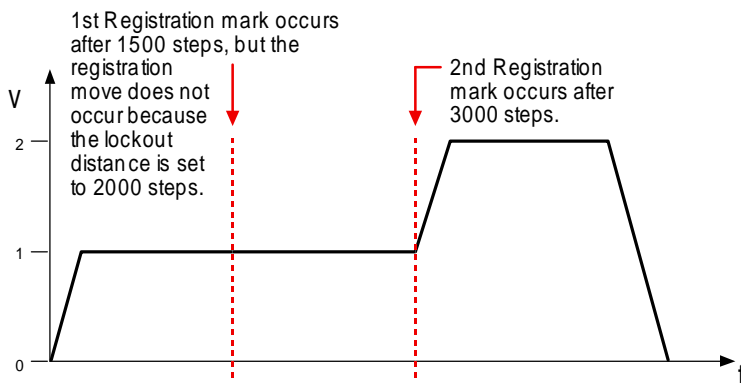
To check the status of the registration input:

```
> !TTRIG.1        Check to see of trigger interrupt input 1A has been activated
  *0              Indicates registration move has not happened on any axis
```

REGSS   Registration Single-Shot

Type	Registration		**Product**	**Rev**
Syntax	`<!><@>REGSS<b><b><b><b><b><b><b><b>`		6K	5.0
Units	n/a			
Range	b=0 (Disable), 1 (Enable), or x (don't care)			
Default	0			
Response	REGSS: *REGSS0000_0000			
	1REGSS: *1REGSS0			
See Also	RE, REG, REGLOD			

The Registration Single Shot (REGSS) command sets the registration such that only one registration move will take place for the specified axis. This allows the user to prevent any other trigger from interrupting the registration move in progress. A GO command will reset the "one shot" condition.

Example – Option A：
A user has a line of material with randomly spaced registration marks. It is known that the first mark must initiate a registration move, and that each registration move cannot be interrupted or the end product will be destroyed. Since the distance between marks is random, it is impossible to predict if a second registration mark will occur before the first registration move has finished.

```
DEL REGI2          ; Delete program (in case program already resides in memory)
DEF REGI2          ; Begin program definition
INFNC1-H           ; Trigger capture mode for trigger 1A on axis 1
RE1                ; Enable registration
V2                 ; Set registration move to a velocity of 2 rps
AD.5               ; a deceleration of 0.5 rev/sec/sec
1REGA20000         ; and a distance of 20000 steps
MC1                ; Start a mode continuous
V1                 ; move at a velocity of 1 rps
GO1                ; Initiate motion
END                ; End program definition
```

Registration Profile:
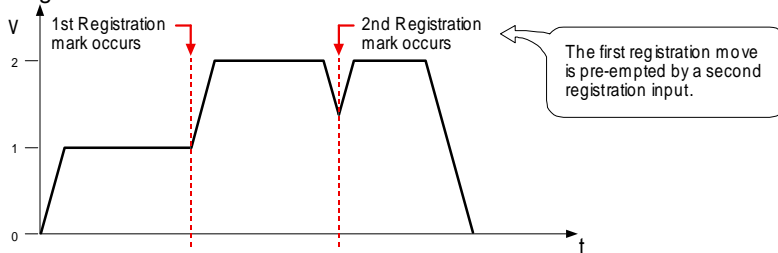
Example – Option B (introducing "single-shot" registration)：
In order to stop the second registration from occurring, REGSS can be used:

```
DEL REGI2b         ; Delete program (in case program already resides in memory)
DEF REGI2b         ; Begin program definition
INFNC1-H           ; Trigger capture mode for trigger 1A on axis 1
RE1                ; Enable registration
V2                 ; Set registration move to a velocity of 2 rps
1REGA20000         ; and a distance of 20000 steps
REGSS1             ; Enable registration single shot mode
MC1                ; Start a mode continuous
V1                 ; move at a velocity of 1 rps
GO1                ; Initiate motion
END                ; End program definition
```

Registration Profile:

www.comoso.com

REPEAT   Repeat Statement

			Product	**Rev**
Type	Program Flow Control or Conditional Branching		6K	5.0
Syntax	`<!>REPEAT`			
Units	n/a			
Range	n/a			
Default	n/a			
Response	n/a			
See Also	JUMP, UNTIL			

The Repeat Statement (REPEAT) command, in conjunction with the UNTIL command, provide a means of conditional program flow. The REPEAT command marks the beginning of the conditional statement. The commands between the REPEAT and the UNTIL command are executed at least once. Upon reaching the UNTIL command, the expression contained within the UNTIL command is evaluated. If the expression is false, the program flow is redirected to the first command after the REPEAT command. If the expression is true, the first command after the UNTIL command is executed.

Up to 16 levels of REPEAT ... UNTIL( ) commands may be nested.

NOTE: Be careful about performing a GOTO between REPEAT and UNTIL. Branching to a different location within the same program will cause the next REPEAT statement encountered to be nested within the previous REPEAT statement, unless an UNTIL command has already been encountered. The JUMP command should be used in this case.

All logical operators (AND, OR, NOT), and all relational operators (=, >, >=, <, <=, <>) can be used within the UNTIL expression. There is no limit on the number of logical operators, or on the number of relational operators allowed within a single UNTIL expression.

The limiting factor for the UNTIL expression is the command length. The total character count for the UNTIL command and expression cannot exceed 80 characters. For example, if you add all the letters in the UNTIL command and the letters within the () expression, including the parentheses and excluding the spaces, this count must be less than or equal to 80.

All assignment operators (A, AD, AS, D, ER, IN, INO, LIM, MOV, OUT, PC, PCE, PCME, PCMS, PE, PER, SS, TIM, US, V, VEL, etc.) can be used within the UNTIL() expression.

Example:
```
REPEAT             ; Beginning of REPEAT ... UNTIL( ) loop
GO1110             ; Initiate motion on axes 1, 2, and 3
VAR1=VAR1+1        ; Increment variable 1 by 1
UNTIL(VAR1=12)     ; Repeat loop until variable 1 = 12
```

RESET   Reset

Type	`Communication Interface`	**Product**	**Rev**
Syntax	`<!>RESET`	6K	5.0
Units	`n/a`		
Range	`n/a`		
Default	`n/a`		
Response	`RESET:   ((power-up message is displayed))`		
See Also	`STARTP, TSTAT`		

The Reset (`RESET`) command affects the 6K controller the same as cycling power. The controller's programs and variables are retained in non-volatile memory; however, all previously entered command values (not saved in programs or variables) will be reset to factory default values.

NOTE: After sending the `RESET` or `!RESET` command to the 6K product, you must wait until you see the power-up message (actual time varies by product) before communicating with the product.

CAUTION: The `RESET` command will disconnect an Ethernet connection.

RUN   Begin Executing a Program

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

The Begin Executing a Program (`RUN`) command executes a program defined with the `DEF` command. A program name consists of 6 or fewer alpha-numeric characters. The `RUN` command can be used inside a program or subroutine. The program can also be run by specifying the name of the program without the `RUN` command. The `RUN` command functions similar to a `GOSUB` command in that control returns to the original program when the called program finishes.

Example:
```
DEF pick        ; Begin definition of program named pick
GO1100          ; Initiate motion on axes 1 and 2
END             ; End program definition
RUN pick        ; Executes program named pick
pick            ; Executes program named pick
```

S   Stop Motion

Type	`Motion`		
Syntax	`<!>S<b><b><b><b><b><b><b><b>`	**Product**	**Rev**
Units	`n/a`	6K	5.0
Range	`b = 0 (do not stop) or 1 (stop)`		
Default	`1`		
Response	`>!S:  No response, instead motion is stopped on all axes.`		
See Also	`C, COMEXC, COMEXS, GO, K`		

The Stop Motion (`S`) command instructs the motor to stop motion on the specified axes. If the Stop (`S`) command is used without any arguments, motion will be stopped on all axes. The Stop command will bring the specified axes to rest using the last deceleration value (`AD`) entered.

NOTE

Since all commands are buffered, the next command does not begin until the previous command has finished. This is important because if you place a Stop (`S`) command after a Go (`GO`) command in a program, the Stop command will have no effect. For the Stop command to have an effect within a program, continuous command processing mode (`COMEXC`) must be enabled. If the Stop (`S`) command is to be used external to the program, the immediate command identifier (`!`) must be used.

If `COMEXS` is set to zero, command processing will be terminated when any stop command is issued, or a stop input is activated. If `COMEXS` is set to 1 or 2, a stop command issued for a specific axis will only stop motion on that axis and will not clear the command buffer. If `COMEXS` is set to 2, a stop command or input will stop motion and clear the command buffer.

If motion is to be paused and later resumed, the stop command must be used without any arguments (`S` or `!S`), and the continue execution on stop (`COMEXS`) command must be enabled. The continue (`!C`) command can then be used to resume motion.

Example:
```
GO1111   ; Initiate motion on all axes
!S1100   ; Stop motion on axes 1 and 2 (must use "!S" to stop motion in progress)
```

SCALE   Enable/Disable Scale Factors

Type	`Scaling`	**Product**	**Rev**
Syntax	`<!>SCALE<b>`	6K	5.0
Units	`n/a`		
Range	`b = 0 (disable) or 1 (enable)`		
Default	`0`		
Response	`SCALE:  *SCALE0`		
See Also	`DRES, ERES, SCLA, SCLD, SCLMAS, SCLV, SFB, TSTAT`		

Scaling allows you to program acceleration, deceleration, velocity, and position values in units of measure that are appropriate for your application. The `SCALE` command is used to enable or disable scaling (`SCALE1` to enable, `SCALEØ` to disable). When scaling is enabled (`SCALE1`), all entered data is multiplied by the appropriate scale factor:

Type of Motion	Accel/Decel Scaling	Velocity Scaling	Distance Scaling
Standard Point-to-Point Motion	`SCLA`	`SCLV`	`SCLD`
Contouring, Linear Interpolation	`SCLD`	`SCLD`	`SCLD`
Following	`SCLA`	`SCLV`	`SCLD` for follower distances `SCLMAS` for master distances

NOTE: Contouring uses only the `SCLD` value to scale all motion parameters; `SCLA` & `SCLV` are not applicable.

When Should I Define Scaling Factors?

Scaling calculations are performed when a program is defined or downloaded. Consequently, you must enable scaling (SCALE1) and define the scaling factors (SCLD, SCLA, SCLV, SCLMAS) *prior* to defining (DEF), uploading (TPROG), or running (RUN or PRUN) the program.

> RECOMMENDATION: Place the scaling commands at the beginning of your program file, *before* the location of any defined programs. This ensures that the motion parameters in subsequent programs in your program file are scaled correctly. When you use Motion Planner's Setup Generator wizard, the scaling commands are automatically placed in the appropriate location in your program file.

> ALTERNATIVE: Scaling factors could be defined via a terminal emulator *just before* defining or downloading a program. Because scaling command values are saved in battery-backed RAM (remembered until you issue a RESET command), all subsequent program definitions and downloads will be scaled correctly.

> RESTRICTIONS: Scaling commands are not allowed in a program. If there are scaling commands in a program, the controller will report an error message ("COMMAND NOT ALLOWED IN PROGRAM") when the program is downloaded. If you intend to upload a program with scaled motion parameters, be sure to use Motion Planner. Motion Planner automatically uploads the scaling parameters and places them at the beginning of the program file containing the uploaded program from the controller. This assures correct scaling when the program file is later downloaded.

Servo Products

Scaling can be used with encoder or analog input feedback sources. When the scaling commands (SCLA, SCLD, etc.) are executed, they are specific only to the current feedback source selected with the last SFB command.

If your application requires switching between feedback sources for the same axis, then for each feedback source, you must select the feedback source with the appropriate SFB command and issue the scaling factors specific to operating with that feedback source.

For example, if you have two axes and will be switching between encoder and ANI feedback, you should include code similar to the following in your setup program:

```
SFB1,1           ; Select encoder feedback (subsequent scaling
                 ; parameters are specific to encoder feedback)
SCLA4000,4000    ; Program accel/decel in revs/sec/sec
SCLV4000,4000    ; Program velocity in revs/sec
SCLD4000,4000    ; Program distances in revs
SFB2,2           ; Select ANI feedback (subsequent scaling
                 ; parameters are specific to ANI feedback)
SCLA205,205      ; Program accel/decel in volts/sec/sec
SCLV205,205      ; Program velocity in volts/sec
SCLD205,205      ; Program distances in volts
```

Units of Measure without Scaling (Scaling is disabled (SCALE0) as the factory default condition):

- Stepper axes: All distance values entered are in commanded counts (sometimes referred to as *motor steps*), and all acceleration, deceleration and velocity values entered are internally multiplied by the DRES command value.

- Servo axes:

	Units of Measure (per feedback source)	
Motion Attribute	**Encoder**	**Analog Input**
Accel/Decel	Revs/sec/sec *	volts/sec/sec
Velocity	Revs/sec *	volts/sec
Distance	Counts **	Counts **

* All accel/decel & velocity values are internally multiplied by the ERES command value.
** Distance is measured in the counts received from the feedback device.

> Contouring & Linear Interpolated Motion: Path acceleration, velocity, and distance are based on the resolution (DRES for steppers, ERES for servos) of axis 1. If multi-tasking is used, path motion units are based on the resolution of the first (lowest number) axis associated with the task (TSKAX).

SCALING EXAMPLES: Refer to page 16.

SCANP   Scan Compiled PLCP Program

			Product	Rev
Type	`PLC Scan Program`		6K	5.0
Syntax	`<!>SCANP<t>`			
Units	`t = text (name of the PLCP program, or CLR)`			
Range	`t = PLCPi, where i is the number of the desired PLCP program,`			
	`or t = CLR (to clear or stop the scan function)`			
Default	`n/a`			
Response	`n/a`			
See Also	`PLCP, PCOMP, PRUN, PUCOMP, TSCAN`			

Use the `SCANP` command to initiate scanning a specific compiled PLCP program (`PLCPi`). For example, `SCANP PLCP3` initiates scanning the program defined as `PLCP3` (defined with `DEF PLCP3`) and compiled (`PCOMP PLCP3`).

The PLCP program is scanned once per 2 ms system update period. Each scan pass is allotted a 0.5 ms window in the 2 ms system update period in which to complete the scan (refer to the diagram on the right). If the scan takes more than 0.5 ms, the scan will pause and continue where it stopped during the next 2-ms system update period. Conversely, if the scan takes less than 0.5 ms, the remaining processing time is used for normal processing.

To check how much time (in 2 ms increments) the last scan took to complete, issue the `TSCAN` command. For example, if the last PLCP program
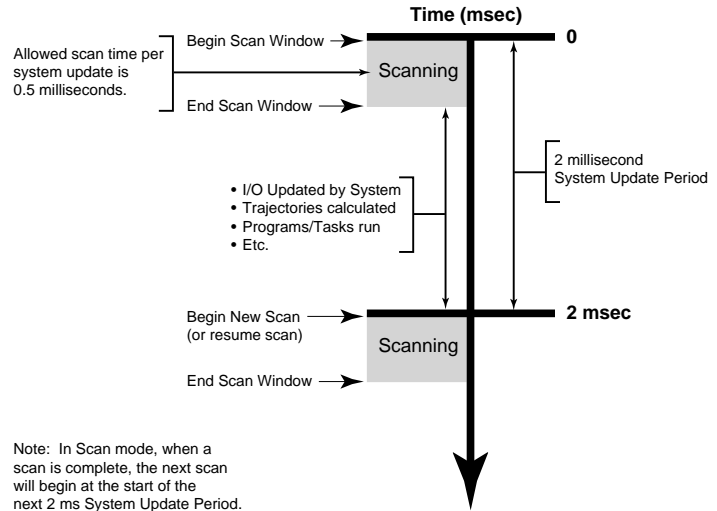
took 3 system updates (2 ms each) to scan, then `TSCAN` would report *TSCAN6, indicating that it took 6 ms to complete the scan.

Launching programs external to the scan: Using the `EXE` command or the `PEXE` command, a scan program can launch another program in a specified task. `EXE` launches a standard, non-compiled program; `PEXE` launches a compiled program.

Stopping the scan: The scan program can be stopped in either of two ways: using the `!K` command, or clearing the scan program by issuing a `SCANP CLR` command.

Timing the PLCP program outputs: It is not possible to control where the PLCP program will pause if the scan takes more than the allowed time. This means that there can be a time lag of several update periods before the outputs, analog outputs, and/or variables affected by the PLCP program are updated. The order in which the scan takes place should be considered when creating PLCP programs to minimize the effects of such a lag. One way to avoid the lag is to create a binary variable as a temporary holding place for the desired output states. The last commands before the `END` statement of the PLCP program can set the outputs according to the final status of the variable, such that all output states are written at the same time, just as the scan completes. This method is demonstrated in the example program below.

For more information on defining a PLC program, refer to the `PLCP` *command description.*

Example:
```
DEF PLCP3
; Binary states of outputs 1-6 are represented by VARB1 bit 1-6.
; Outputs 1-6 are set at the end of the program.
VARB1=b000000          ; Initialize binary variable 1
IF(IN.1=b1)            ; If Input 1 is ON, turn Output 1 ON
VARB1=VARB1 | b100000  ; Set binary bit for output 1 only to ON
NIF
IF(IN.2=b1)            ; If Input 2 is ON, turn Output 2 ON
VARB1=VARB1 | b010000  ; Set binary bit for output 2 only to ON
NIF
IF(IN.3=b1)            ; If Input 3 is ON, turn Output 3 ON
VARB1=VARB1 | b001000  ; Set binary bit for output 3 only to ON
NIF
IF(IN.4=b1)            ; If Input 4 is ON, turn Output 4 ON
VARB1=VARB1 | b000100  ; Set binary bit for output 4 only to ON
NIF
IF(IN.5=b1)            ; If Input 5 is ON, turn Output 5 ON
VARB1=VARB1 | b000010  ; Set binary bit for output 5 only to ON
NIF
IF(IN.6=b1)            ; If Input 6 is ON, turn Output 6 ON
VARB1=VARB1 | b000001  ; Set binary bit for output 6 only to ON
NIF
OUT(VARB1)            ; Turn on appropriate outputs
END

PCOMP PLCP3           ; Compile program PLCP3

SCANP PLCP3           ; Run compiled program PLCP3 in Scan mode
                      ; The diagram below illustrates the scan.
```


Time (msec)

```
DEF PLCP3
VARB1=b000000
IF(IN.1=b1)
VARB1=VARB1 | b100000
NIF
IF(IN.2=b1)
VARB1=VARB1 | b010000
NIF
IF(IN.3=b1)
VARB1=VARB1 | b001000
NIF
IF(IN.4=b1)
VARB1=VARB1 | b000100
NIF
IF(IN.5=b1)
VARB1=VARB1 | b000010
NIF
IF(IN.6=b1)
VARB1=VARB1 | b000001
ELSE
VARB1=VARB1 | b000000
NIF
OUT(VARB1)
END
```

Pause Scan

Scanning — 0

Scanning — 2 msec

Scan Complete

Scanning — 4 msec

Begin New Scan

Scanning — 6 msec

2 System Update Periods are needed to complete the scan
for compiled program PLCP3, for a total of 4 msec.
The response to a TSCAN command would be: *TSCAN4.

SCLA
Acceleration Scale Factor

			Product	**Rev**
Type	`Scaling`		6K	5.0
Syntax	`<!><@><a>SCLA<i>,<i>,<i>,<i>,<i>,<i>,<i>,<i>`			
Units	`i = counts/unit`			
Range	`1 - 999,999`			
Default	`4000`			
	(Servos auto-detect based on SFB: 4000 if encoder, 205 if ANI)			
Response	`SCLA:  *SCLA4000,4000,4000,4000,4000,4000,4000,4000`			
	`1SCLA:  *1SCLA4000`			
See Also	`ANIRNG, FMAXA, SCALE, SCLD, SCLMAS, SCLV, SFB, TSTAT`			

When scaling is enabled (SCALE1), all point-to-point acceleration values (A, AA, HOMA, HOMAA, JOGA, JOGAA, JOYA, JOYAA) and deceleration values (AD, ADA, LHAD, LHADA, LSAD, LSADA, HOMAD, HOMADA, JOGAD, JOGADA, JOYAD, JOYADA) are multiplied by the Acceleration Scale Factor (SCLA) command. Since the units are counts/unit, and all the acceleration values are in units/sec/sec, all accelerations will thus be internally represented as counts/sec/sec.

Stepper axes: If scaling is enabled (SCALE1), the entered accel and decel values are internally multiplied by the acceleration scaling factor (SCLA) to convert user units/sec/sec to commanded counts/sec/sec (sometimes referred to as "motor steps"/sec/sec). The entered values are always in reference to commanded counts, regardless of the existence of an encoder.

If scaling is disabled (SCALEØ), all accel and decel values are entered in commanded revs/sec/sec; these values are internally multiplied by the drive resolution (DRES) value to obtain accel and decel values in commanded counts/sec/sec for the motion trajectory calculations.

Servo axes: If scaling is enabled (SCALE1), the entered accel and decel values are internally multiplied by the acceleration scaling factor (SCLA) to convert user units/sec/sec to encoder or ANI counts/sec/sec.

If scaling is disabled (SCALEØ), all accel and decel values are entered in encoder revs/sec/sec or ANI volts/sec/sec; encoder values are internally multiplied by the encoder resolution (ERES) value to obtain accel and decel values in counts/sec/sec for the motion trajectory calculations.

As the acceleration scaling factor (SCLA) changes, the resolution of the acceleration and deceleration values and the number of positions to the right of the decimal point also change (see table at right). An acceleration value with greater resolution than allowed will be truncated. For example, if scaling is set to SCLA1Ø, the A9.9999 command would be truncated to A9.9.

SCLA **(counts/unit)**	**Decimal Places**
1 - 9	0
10 - 99	1
100 - 999	2
1000 - 9999	3
10000 - 99999	4
100000 - 999999	5

The following equations can help you determine the range of acceleration and deceleration values.

Axis Type	Min. Accel or Decel (resolution)		Max. Accel or Decel	
Stepper	$\dfrac{0.001 * DRES}{SCLA}$		$\dfrac{999.9999 * DRES}{SCLA}$	
Servo	Encoder Feedback:	$\dfrac{0.001 * ERES}{SCLA}$	Encoder Feedback:	$\dfrac{999.9999 * ERES}{SCLA}$
	ANI Feedback: *	$\dfrac{0.8205}{SCLA}$	ANI Feedback: *	$\dfrac{20479.9795}{SCLA}$

* This calculation assumes the analog input range (ANIRNG value) is left in its default setting (range is -10V to +10V).

MORE ABOUT SCALING

For additional details on scaling, including scaling examples, refer to page 16.

SCLD   Distance Scale Factor

Type	Scaling	**Product**	**Rev**
Syntax	`<!><@><a>SCLD<i>,<i>,<i>,<i>,<i>,<i>,<i>,<i>`	6K	5.0
Units	`i = counts/unit`		
Range	1 - 999,999		
Default	1		
	(Servos auto-detect based on SFB: 1 if encoder, 205 if ANI)		
Response	SCLD: *SCLD1,1,1,1,1,1,1,1		
	1SCLD: *1SCLD1		
See Also	[ANI], D, [FB], FOLRN, FSHFD, [PANI], [PC], [PCC], [PCE], [PCMS], [PE], [PER], PSET, [PSHF], [PSLV], REG, REGLOD, SCALE, SCLA, SCLV, SCLMAS, SFB, SMPER, TANI, TFB, TPANI, TPC, TPCC, TPCE, TPCMS, TPE, TPER, TPSHF, TPSLV, TSTAT		

If scaling is enabled (SCALE1), all D, PSET, SMPER, and REG command values are internally multiplied by the Distance Scale Factor (SCLD) value. Since the SCLD units are in terms of counts/unit, all distances will thus be internally represented in counts. For instance, if your distance scaling factor is 10000 (SCLD10000) and you enter a distance of 75 (D75), the actual distance moved will be 750,000 (10000 x 75) counts.

This command is useful for allowing the user to specify distances in any unit. For example, if the user had a 25000 step/revolution drive and wanted distance units in terms of revolutions, then SCLD should be set to 25000, and scaling should be enabled (SCALE1).

As the distance scaling factor (SCLD) changes, the resolution of all distance commands and the number of positions to the right of the decimal point also change (see table below). A distance value with greater resolution than allowed will be truncated (e.g., if scaling is set to SCLD25000, the D1.99999 command would be truncated to D1.9999).

SCLD (counts/unit)	Distance Resolution (units)	Distance Range (units)	Decimal Places
1 - 9	1	0 - ±999,999,999	0
10 - 99	0.1	0.0 - ±99,999,999.9	1
100 - 999	0.01	0.00 - ±9,999,999.99	2
1000 - 9999	0.001	0.000 - ±999,999.999	3
10000 - 99999	0.0001	0.0000 - ±99,999.9999	4
100000 - 999999	0.00001	0.00000 - ±9999.99999	5

FRACTIONAL STEP TRUNCATION

If you are operating in the preset positioning mode (MC0), when the distance scaling factor (SCLD) and the distance value are multiplied, a fraction of one step may possibly be left over. This fraction is truncated when the distance value is used in the move algorithm. This truncation error can accumulate over a period of time, when performing incremental moves continuously in the same direction. To eliminate this truncation problem, set the distance scale factor (SCLD) to 1, or a multiple of 10.

MORE ABOUT SCALING

For additional details on scaling, including scaling examples, refer to page 16.

SCLMAS   Master Scale Factor

Type	Following; Scaling	**Product**	**Rev**
Syntax	`<!><@><a>SCLMAS<i>,<i>,<i>,<i>,<i>,<i>,<i>,<i>`	6K	5.0
Units	i = scaling factor		
Range	i = 1 – 999999		
Default	1		
Response	`SCLMAS    *SCLMAS1,1,1,1,1,1,1,1`		
	`1SCLMAS   *1SCLMAS1`		
See Also	FMCLEN, FMCP, FOLEN, FOLMD, FOLRD, FOLRN, GOWHEN, [PCMS],		
	[PMAS], SCALE, SCLD, TPMAS, TPCMS		

The Master Scale Factor (SCLMAS) command internally multiplies all Following master values by the specified scale factor value. Since the SCLMAS units are in terms of counts/unit, all distances will thus be internally represented in counts. For instance, if your master scaling factor is 10000 (SCLMAS10000) and you enter a master parameter of 75 (e.g., FOLMD75), the internal value will be 750,000 (10000 x 75) counts.

NOTE: The SCLMAS command will not take effect unless scaling is enabled (SCALE1).

This command allows you to specify distances in any unit. For example, if you had a 4000 step/revolution encoder as the master and wanted master units in terms of revolutions, then SCLMAS should be set to 4000.

As the master scaling factor (SCLMAS) changes, the resolution of all master parameter values and the number of positions to the right of the decimal point also change (see table below). A master parameter value with greater resolution than allowed will be truncated (e.g., if scaling is set to SCLD4000, the FOLMD1.9999 command would be truncated to FOLMD1.999).

SCLMAS (counts/unit)	Master Resolution (units)	Master Range (units)	Decimal Places
1 - 9	1	0 - ±999,999,999	0
10 - 99	0.1	0.0 - ±99,999,999.9	1
100 - 999	0.01	0.00 - ±9,999,999.99	2
1000 - 9999	0.001	0.000 - ±999,999.999	3
10000 - 99999	0.0001	0.0000 - ±99,999.9999	4
100000 - 999999	0.00001	0.00000 - ±9999.99999	5

FRACTIONAL STEP TRUNCATION

If you are specifying master distance values (FOLMD), when the master scaling factor (SCLMAS) and the distance value are multiplied, a fraction of one count may possibly be left over. This fraction is truncated when the distance value is used in the move algorithm. This truncation error can accumulate when performing several moves over the specified master distance. To eliminate this truncation problem, set the master scale factor (SCLMAS) to 1, or a multiple of 10.

Example: (refer also to the FOLEN examples, and page 16)

The commands below are a subset of the set-up parameters for an application in which axis 1 is following the encoder input on axis #3 at a 1-to-1 ratio.

```
SCALE1              ; Enable parameter scaling
SCLA25000           ; Set follower acceleration scale factor to 25000 for axis 1
SCLV25000           ; Set follower velocity scale factor to 25000 for axis 1
SCLD25000           ; Set follower distance scale factor to 25000 for axis 1
SCLMAS4000          ; Set master scale factor to 4000 for axis 1
FOLMAS31            ; Axis 1 using encoder input #3 as master
FOLRN1              ; Set follower-to-master Following ratio numerator to 1
                    ; (scaled by SCLD)
FOLRD1              ; Set follower-to-master Following ratio denominator to 1.
                    ; This sets the ratio to 1:1 (scaled the SCLMAS).
                    ; The actual ratio in counts = 25000 to 4000 = 6.25 follower
                    ; axis counts per master count.
```

SCLV   Velocity Scale Factor

Type	`Scaling`	**Product**	**Rev**
Syntax	`<!><@><a>SCLV<i>,<i>,<i>,<i>,<i>,<i>,<i>,<i>`	6K	5.0
Units	`i = counts/unit`		
Range	`1 – 999,999`		
Default	`4000`		
	`(Servos auto-detect based on SFB: 4000 if encoder, 205 if ANI)`		
Response	`SCLV:   *SCLV4000,4000,4000,4000 ...`		
	`1SCLV:  *1SCLV4000`		
See Also	`ANIRNG, FMAXV, HOMV, HOMVF, JOGVH, JOGVL, JOYVH, JOYVL, SCALE,`		
	`SCLA, SCLD, SFB, TSTAT, V`		

When scaling is enabled (SCALE1), all velocity values (HOMV, HOMVF, JOGVH, JOGVL, JOYVH, JOYVL, V)
are multiplied by the Velocity Scale Factor (SCLV) command. Since the units are counts/unit, all velocities
will thus be internally represented in counts/sec.

Steppers: If scaling is enabled (SCALE1), the entered velocity values are internally multiplied by SCLV to
convert user units/sec to commanded counts/sec.

If scaling is disabled (SCALEØ), all velocity values are entered in commanded revs/sec; these
values are internally multiplied by the drive resolution (DRES) value to obtain velocity values in
commanded counts/sec for the motion trajectory calculations.

Servos: If scaling is enabled (SCALE1), the entered velocity values are internally multiplied by SCLV to
convert user units/sec to encoder or ANI counts/sec.

If scaling is disabled (SCALEØ), all velocity values are entered in encoder revs/sec or ANI
volts/sec; encoder values are internally multiplied by the encoder resolution (ERES) value to
obtain velocity values in counts/sec for the motion trajectory calculations.

As the velocity scaling factor (SCLV) changes, the resolution of the velocity commands and the number of
positions to the right of the decimal point also change (see table below). A velocity value with greater
resolution than allowed will be truncated. For example, if scaling is set to SCLV1Ø, the V1.9999 command
would be truncated to V1.9.

SCLV (steps/unit)	Velocity Resolution (units/sec)	Decimal Places
1 - 9	1	0
10 - 99	0.1	1
100 - 999	0.01	2
1000 - 9999	0.001	3
10000 - 99999	0.0001	4
100000 - 999999	0.00001	5

Use the following equations to determine the maximum velocity range for your product type.

	Max. Velocity for Stepper Axes	**Max. Velocity for Servo Axes** (Servos: determined by feedback source selected for axis #1)	
$\dfrac{n}{SCLV}$	n = maximum velocity is determined by the PULSE command setting.	Encoder Feedback:	$\dfrac{6{,}500{,}000}{SCLV}$
		ANI Feedback: *	$\dfrac{1000 * 205}{SCLV}$

* This calculation assumes the analog input range (ANIRNG value) is left in its default setting (range is -10V to +10V).

MORE ABOUT SCALING

For additional details on scaling, including scaling examples, refer to page 16.

[SEG]   Number of Free Segment Buffers

Type	Compiled Motion; 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	MEMORY, [SS], TDIR, TMEM, TSEG, TSS		

Use the SEG operator to assign the number of free memory segment buffers in *compiled memory* to a variable (VAR), or to make a comparison against another value. "Compiled memory" is the partition of the 6K controller's non-volatile memory that stores compiled profiles & PLC programs. Compiled profiles/programs could be a multi-axis *contour* (a series of arcs and lines), an *individual axis profile* (a series of GOBUF commands), a *compound profile* (combination of multi-axis contours and individual axis profiles), or a *PLC program* (for PLC Scan Mode).

System status bit (see TSSF, TSS, and SS) 29 to set when the compiled memory is 75% full, and bit 30 is set if the compiled memory is 100% full.

Syntax: VARn=SEG where "n" is the variable number,
or SEG can be used in an expression such as IF(SEG=1)

SFB   Select Servo Feedback Source

Type	Controller Configuration or Servo	**Product**	**Rev**
Syntax	<@><a>SFB<i>,<i>,<i>,<i>,<i>,<i>,<i>,<i>	6K	5.0
Units	i = feedback source identifier		
Range	i = 0 (open loop, disable gains), 1 (encoder), or 2 (ANI input)	(applicable to servo axes only)	
Default	1		
Response	SFB *SFB1,1,1,1,1,1,1,1		
	1SFB *1SFB1		
See Also	[ANI], ANIFB, ANIRNG, AXSDEF, ERES, [FB], OUTPn, [PANI], [PCE], [PE], PSET, SCALE, SCLD, SGAF, SGI, SGILIM, SGP, SGV, SGVF, SMPER, SOFFS, TANI, TFB, TPANI, TPE		

Use the SFB command to select the servo feedback source to be used by each axis. The options are:

Options	Physical Location	Measurement*	Resolution Command
1—Encoder	**ENCODER** connector only	Encoder counts	ERES (default is 4000 counts/rev)
2—Analog ("ANI") input **	Analog input SIM on external I/O brick	ADC counts	ANIRNG (default is 205 counts/volt)

* With scaling enabled (SCALE1), encoder and ANI feedback is scaled by the SCLD value.
** Before an analog input can be selected for feedback, it must be configured with the ANIFB command.

NOTE

Parameters for scaling (SCLA, SCLD, etc.) , tuning gains (SGI, SGP, etc.), position offset (PSET) and maximum position error (SMPER) are specific to the feedback source currently selected with the last SFB command.

If your application requires switching between feedback sources for the same axis, then for each feedback source, you must issue the SFB command and then enter the scaling, gains, PSET and SMPER commands specific to that feedback source.

The feedback source can be changed only if motion is not in progress. When the feedback source is changed, the new setpoint will be determined by taking the new feedback source's value and adding any existing position error. Changing the source will disable the Output On Position commands (OUTPn).

Setting the SFB command value to zero has these effects:

- **WARNING**: The end-of-travel limits are disabled. Make sure that it is safe to operate without end-of-travel limits before using SFBØ.

- Gain values (SGILIM, SGAF, SGI, SGP, etc.) set to zero (open-loop operation).

- SMPER value set to zero (position error is allowed to increase without causing a fault.

- Subsequent attempts to change gain values or SMPER will cause an error message ("NOT ALLOWED IF SFBØ")

- SOFFS set to zero, but allows subsequent servo offset changes to affect motion.

- Disables output-on-position (OUTPA - OUTPH) functions.

- Any subsequent changes to PSET, PSETCLR, SCLD, SCLA, SCLV, and SOFFS are lost when another feedback source is selected.

Recommendation: Use the Disable Drive On Kill more, enabled with the KDRIVE command, so that the controller will shut down the drive if a kill command (e.g., !K) is executed or if a kill input is activated. Keep in mind that shutting down the drive allows the load to freewheel if there is not brake installed.

Example (to be placed outside of a program, because of the scaling parameters)**:**

```
DRIVE0          ; Disable (shutdown) axis #1
SFB1            ; Select encoder feedback for axis #1 (subsequent scaling,
                ; gains, and PSET are specific to encoder feedback operation)
ERES4000        ; Set encoder resolution
SCLA4000        ; Set scaling for programming acceleration in revs/sec/sec
SCLV4000        ; Set scaling for programming velocity in revs/sec
SCLD4000        ; Set scaling for programming distance in revs
SGP5            ; Set proportional feedback gain to 5
SGI1            ; Set integral feedback gain to 1
SGV1            ; Set velocity feedback gain to 1
PSET0           ; Set current position as absolute position zero
1ANIRNG.17=4    ; Select a voltage range of –10V to +10V for the 1st analog
                ; input channel in SIM slot 3 (I/O location 17) of I/O brick 1.
                ; This means the counting resolution will be 205 counts/volt.
ANIFB1-17       ; Select the 1st analog input channel in SIM slot 3
                ; (I/O location 17) of I/O brick 1 to be used as
                ; feedback for axis 1.  ((required before the SFB command))
SFB2            ; Select ANI feedback for axis #1 (subsequent scaling, gains,
                ; and PSET are specific to ANI feedback operation)
SCLA205         ; Set scaling for programming acceleration in volts/sec/sec
SCLV205         ; Set scaling for programming velocity in volts/sec
SCLD205         ; Set scaling for programming distance in volts
SGP1            ; Set proportional feedback gain to 1
SGI0            ; Set integral feedback gain to zero
SGV.5           ; Set velocity feedback gain to 0.5
PSET0           ; Set current position as absolute position zero
SFB1            ; Select encoder feedback for axis #1
```

SGAF   Acceleration Feedforward Gain

Type	`Servo`	**Product**	**Rev**
Syntax	`<!><@><a>SGAF<r>,<r>,<r>,<r>,<r>,<r>,<r>,<r>`	6K	5.0
Units	`r = microvolts/step/sec/sec`		
Range	`0.00000000 - 2800000.00000000`	(applicable to servo	
Default	`0`	axes only)	
Response	`SGAF:    *SGAF0,0,0,0,0,0,0,0`		
	`1SGAF:   *1SGAF0`		
See Also	`SFB, SGENB, SGI, SGP, SGSET, SGV, SGVF, TGAIN, TSGSET`		

Use the Acceleration Feedforward Gain (`SGAF`) command to set the gain for the acceleration feedforward term in the servo control algorithm. Introducing acceleration feedforward control improves *position tracking performance* when the system is commanded to accelerate or decelerate.

The `SGAF` value is multiplied by the *commanded acceleration* (calculated by the 6K controller's DSP move profile routine) to produce the control signal.

Acceleration feedforward control can improve the performance of contouring and linear interpolation applications, as well as reduce the time required to reach the commanded velocity. *However, if your application only requires point-to-point moves, acceleration feedforward control is not necessary (leave the SGAF command setting at zero—default).*

Acceleration feedforward control does not affect the servo system's stability, nor does it have any effect at constant velocity or at steady state.

NOTE

The `SGAF` command is specific to the current feedback source (selected with the last `SFB` command). Therefore, if your application requires switching between feedback sources for the same axis, then for each feedback source, you must select the feedback source with the appropriate `SFB` command and then issue the `SGAF` command with the gain values specific to the selected feedback source.

For more information on servo tuning and how the acceleration feedforward gain affects performance, refer to your product's *Installation Guide* or to the *Servo Tuner User Guide*.

Example:
```
SGAF0.5555,43.554,0,0      ; Set the acceleration feedforward for axes 1 and 2
```

SGENB   Enable a Servo Gain Set

Type	`Servo`	**Product**	**Rev**
Syntax	`<!><@><a>SGENB<i>,<i>,<i>,<i>,<i>,<i>,<i>,<i>`	6K	5.0
Units	`i = gain set identification number (see SGSET command)`		
Range	`1 - 5`	(applicable to servo	
Default	`n/a`	axes only)	
Response	`n/a`		
See Also	`SFB, SGAF, SGAFN, SGI, SGILIM, SGP, SGSET, SGV, SGVF, SOFFS, TGAIN, TSGSET`		

This command allows you to enable any combination of the five gain sets to any combination of axes. The gain sets are set with the `SGSET` command. A gain set can be enabled during motion at any specified point in the profile, or when not in motion. For example, you could use one set of gain parameters for the constant velocity portion of the profile, and when you approach the target position a different set of gains can be enabled.

NOTE

The tuning gains in a given gain set are specific to the feedback source that was in use (selected with the last `SFB` command) at the time the gains were established with the respective gain commands (`SGI`, `SGP`, etc.). Make sure that the gain set you enable is appropriate to the feedback source you are using at the time.

For more information on servo tuning, refer to your product's *Installation Guide* or to the Motion Planner help system.

Example:
```
SGP5,5,10,10   ; Sets the gains for the proportional gain
SGI.1,.1,0,0   ; Sets the gains for the integral gain
SGV50,60,0,0   ; Sets the gains for the velocity gain
SGVF5,6,10,11   ; Sets the gains for the velocity feedforward gain
SGAF0,0,0,0   ; Sets the gains for the acceleration feedforward gain
SGSET3   ; Assign SGP, SGI, SGV, SGVF, & SGAF gains to servo gain set 3
SGP75,75,40,40 ; Sets the gains for the proportional gain
SGI5,5,5,7   ; Sets the gains for the integral gain
SGV1,.45,2,2   ; Sets the gains for the velocity gain
SGVF0,8,0,9   ; Sets the gains for the velocity feedforward gain
SGAF18,20,22,24 ; Sets the gains for the acceleration feedforward gain
SGSET1   ; Assign SGP, SGI, SGV, SGAF, & SGVF gains to servo gain set 1
SGENB1,3,3,1   ; Enables gain set 1 gains on axis 1 &4; enables gain set 3 on
                ; axis 2 & 3
TGAIN   ; Displays the current value for all gains. Example response:
                ;   *SGP75,5,10,40
                ;   *SGI5,.1,0,7
                ;   *SGV1,60,0,2
                ;   *SGVF0,6,10,9
                ;   *SGAF18,0,0,24
```

---

# SGI          Integral Feedback Gain

| | | | |
|---|---|---|---|
| Type | Servo | **Product** | **Rev** |
| Syntax | <!><@><a>SGI<r>,<r>,<r>,<r>,<r>,<r>,<r>,<r> | 6K | 5.0 |
| Units | r = millivolts/step * sec | | |
| Range | 0.00000000–2,800,000.00000000 | (applicable to servo | |
| Default | 0.0 | axes only) | |
| Response | SGI:    *SGI0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0 | | |
| | 1SGI:   *1SGI0.0 | | |
| See Also | SFB, SGAF, SGENB, SGILIM, SGP, SGSET, SGV, SGVF, TGAIN, TSGSET | | |

Use the Integral Gain (SGI) command to set the gain of the integral term in the control algorithm. The primary function of the integral gain is to reduce or eliminate final position error (e.g., due to friction, gravity, etc.) and improve system accuracy during motion. If a position error exists (commanded position not equal to actual position—see TPER command), this control signal will ramp up until it is high enough to overcome the friction and drive the motor toward its commanded position. *If acceptable position accuracy is achieved with proportional gain (SGP), then the integral gain (SGI) need not be used.*

If the integral gain is set too high relative to the other gains, the system may become oscillatory or unstable. The integral gain can also cause excessive position overshoot and oscillation if an appreciable position error has persisted long enough during the transient period (time taken to reach the position setpoint); this effect can be reduced by using the SGILIM command to limit the integral term windup.

---

**NOTE**

The SGI command is specific to the current feedback source (selected with the last SFB command). Therefore, if your application requires switching between feedback sources for the same axis, then for each feedback source, you must select the feedback source with the appropriate SFB command and then issue the SGI command with the gain values specific to the selected feedback source.

---

For more information on servo tuning, refer to your product's *Installation Guide* or to the Motion Planner help system.

**Example:**
```
SGI15,14.5   ; Set the integral gain for axes 1 and 2
```

## SGILIM          Integral Windup Limit

| | | | |
|---|---|---|---|
| Type | `Servo` | **Product** | **Rev** |
| Syntax | `<!><@><a>SGILIM<r>,<r>,<r>,<r>,<r>,<r>,<r>,<r>` | 6K | 5.0 |
| Units | `r = volts` | | |
| Range | `0-65,535` | | |
| Default | `200` | (applicable to servo axes only) | |
| Response | `SGILIM:    *SGILIM200,200,200,200,200,200,200,200` | | |
| | `1SGILIM:  *1SGILIM200` | | |
| See Also | `SFB, SGENB, SGI, TGAIN, TSGSET` | | |

If integral control (`SGI`) is used and an appreciable position error has persisted long enough during the transient period (time taken to reach the setpoint), the control signal generated by the integral action can end up too high and saturate to the maximum level of the controller's analog control signal output. This phenomenon is called *integrator windup*.

After windup occurs, it will take a while before the integrator output returns to a level within the limit of the controller's output. Such a delay causes excessive position overshoot and oscillation. Therefore, the integral windup limit (`SGILIM`) command is provided for you to set the absolute limit of the integral and, in essence, turn off the integral action as soon as it reaches the limit; thus, position overshoot and oscillation can be reduced.

---
**NOTE**

The `SGILIM` command is specific to the current feedback source (selected with the last `SFB` command). Therefore, if your application requires switching between feedback sources for the same axis, then for each feedback source, you must select the feedback source with the appropriate `SFB` command and then issue the `SGILIM` command with the gain values specific to the selected feedback source.

---

For more information on servo tuning, refer to your product's *Installation Guide* or to the Motion Planner help system.

**Example:**
```
SGI44,43,55,0   ; Sets the integral gain term
SGILIM15,15,15,15 ; Sets the integral windup limit on the integral gain term
```

## SGP          Proportional Feedback Gain

| | | | |
|---|---|---|---|
| Type | `Servo` | **Product** | **Rev** |
| Syntax | `<!><@><a>SGP<r>,<r>,<r>,<r>,<r>,<r>,<r>,<r>` | 6K | 5.0 |
| Units | `r = millivolts/step` | | |
| Range | `0.00000000-2,800,000.00000000` | (applicable to servo axes only) | |
| Default | `0.5` | | |
| Response | `SGP:   *SGP0.5,0.5,0.5,0.5 ...` | | |
| | `1SGP:  *1SGP0.5` | | |
| See Also | `SFB, SGAF, SGENB, SGI, SGSET, SGV, SGVF, TGAIN, TSGSET` | | |

This command allows you to set the gain of the proportional term in the servo control algorithm. The output of the proportional term is proportional to the difference between the commanded position and the actual position read from the feedback device. The primary function of the proportional term is to stabilize the system and speed up the response. It can also be used to reduce the steady state position error.

When the proportional gain (`SGP`) is used alone (i.e., the other gain terms are set to zero), setting this gain too high can cause the system to become oscillatory, underdamped, or even unstable.

---
**NOTE**

The `SGP` command is specific to the current feedback source (selected with the last `SFB` command). Therefore, if your application requires switching between feedback sources for the same axis, then for each feedback source, you must select the feedback source with the appropriate `SFB` command and then issue the `SGP` command with the gain values specific to the selected feedback source.

---

For more information on servo tuning, refer to your product's *Installation Guide* or to the Motion Planner help system.

**Example:**
```
SGP10,4.22233,2.22,.0445245   ; Sets the proportional gain of all axes
```

## SGSET          Save a Servo Gain Set

| | | | |
|---|---|---|---|
| Type | Servo | **Product** | **Rev** |
| Syntax | <!>SGSET<i> | 6K | 5.0 |
| Units | i = gain set identification number | | |
| Range | 1-5 | (applicable to servo | |
| Default | n/a | axes only) | |
| Response | n/a | | |
| See Also | SFB, SGAF, SGENB, SGI, SGILIM, SGP, SGV, SGVF, SOFFS, TGAIN, TSGSET | | |

This command allows you to save the presently assigned gain values (SGP, SGI, SGV, SGAF, and SGVF) as a set of gains. Stand-alone servo controllers save (into battery-backed RAM) the gains and the axes and feedback sources to which they are assigned. Up to 5 sets of gains can be saved. Any gain set can be displayed using the TSGSET command.

Any gain set can be enabled with the SGENB command during motion at any specified point in the profile, or when not in motion. For example, you could use one set of gain parameters for the constant velocity portion of the profile, and when you approach the target position a different set of gains can be enabled.

---

**NOTE**

The tuning gains in a given gain set are specific to the feedback source that was in use (selected with the last SFB command) at the time the gains were established with the respective gain commands (SGI, SGP, etc.). If your application requires you to switch between feedback sources for the same axis, make sure that the gain set you enable is appropriate to the feedback source you are using at the time.

---

For more information on servo tuning, refer to your product's *Installation Guide* or to the Motion Planner help system.

**Example:**
```
SGP5,5,10,10   ; Sets the gains for the proportional gain
SGI.1,.1,0,0   ; Sets the gains for the integral gain
SGV50,60,0,0   ; Sets the gains for the velocity gain
SGVF5,6,10,11   ; Sets the gains for the velocity feedforward gain
SGAF0,0,0,0   ; Sets the gains for the acceleration feedforward gain
SGSET3   ; Assign SGP, SGI, SGV, SGVF, & SGAF gains to servo gain set 3
SGP75,75,40,40 ; Sets the gains for the proportional gain
SGI5,5,5,7   ; Sets the gains for the integral gain
SGV1,.45,2,2   ; Sets the gains for the velocity gain
SGVF0,8,0,9   ; Sets the gains for the velocity feedforward gain
SGAF18,20,22,24 ; Sets the gains for the acceleration feedforward gain
SGSET1   ; Assign SGP, SGI, SGV, SGAF, & SGVF gains to servo gain set 1
SGENB1,3,3,1   ; Enables gain set 1 gains on axis 1 &4; enables gain set 3 on
                ; axis 2 & 3
TGAIN   ; Displays the current value for all gains. Example response:
                ;   *SGP75,5,10,40
                ;   *SGI5,.1,0,7
                ;   *SGV1,60,0,2
                ;   *SGVF0,6,10,9
                ;   *SGAF18,0,0,24
```

# SGV          Velocity Feedback Gain

| | | | |
|---|---|---|---|
| Type | Servo | **Product** | **Rev** |
| Syntax | `<!><@><a>SGV<r>,<r>,<r>,<r>,<r>,<r>,<r>,<r>` | 6K | 5.0 |
| Units | r = microvolts/step/sec | | |
| Range | 0.00000000–2,800,000.00000000 | | |
| Default | 0.0 | (applicable to servo | |
| Response | SGV:  *SGV0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0 | axes only) | |
| | 1SGV: *1SGV0.0 | | |
| See Also | ERES, SFB, SGAF, SGI, SGP, SGVF, TGAIN, TSGSET | | |

This command allows you to control the velocity feedback gain in the servo algorithm. Using velocity feedback, the controller's output signal is made proportional to the velocity, or rate of change, of the feedback device position. Since it acts on the rate of change of the position, the action of this term is to anticipate position error and correct it before it becomes too large. This increases damping and tends to make the system more stable.

If this term is too large, the response will be slowed to the point that the system is over-damped. This gain can increase position tracking error, which can be countered by the velocity feedforward term (SGVF).

Since the feedback device signal has finite resolution, the velocity accuracy has a limit. Therefore, if the velocity feedback gain (SGV) is too high, the errors due to the finite resolution are magnified and a noisy, or *chattering*, response may be observed.

---

**NOTE**

The SGV command is specific to the current feedback source (selected with the last SFB command). Therefore, if your application requires switching between feedback sources for the same axis, then for each feedback source, you must select the feedback source with the appropriate SFB command and then issue the SGV command with the gain values specific to the selected feedback source.

---

For more information on servo tuning, refer to your product's *Installation Guide* or to the Motion Planner help system.

**Example:**
```
SGV100,97,43.334,0 ; Sets the velocity gain term for all the axes
```

# SGVF          Velocity Feedforward Gain

| | | | |
|---|---|---|---|
| Type | Servo | **Product** | **Rev** |
| Syntax | `<!><@><a>SGVF<r>,<r>,<r>,<r>,<r>,<r>,<r>,<r>` | 6K | 5.0 |
| Units | r = microvolts/step/sec | | |
| Range | 0.00000000–2,800,000.00000000 | | |
| Default | 0.0 | (applicable to servo | |
| Response | SGVF:   *SGVF0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0 | axes only) | |
| | 1SGVF:  *1SGVF0.0 | | |
| See Also | SFB, SGAF, SGENB, SGI, SGP, SGSET, SGV, TGAIN, TSGSET | | |

Use the Velocity Feedforward Gain (SGVF) command to set the velocity feedforward gain. Introducing velocity feedforward control improves *position tracking performance* when the system is commanded to move at constant velocity. The tracking error is mainly attributed to friction, torque load, and velocity feedback control (SGV).

The SGVF value is multiplied by the *commanded velocity* (calculated by the 6K controller's DSP move profile routine) to produce the control signal.

Velocity feedforward control can improve the performance of interpolation (linear and circular) application. *However, if your application only requires short, point-to-point moves, velocity feedforward control is not necessary (leave the SGVF command setting at zero—default).*

Because velocity feedforward control is not in the servo feedback loop, it does not affect the servo system's stability, nor does it have any effect at steady state. Therefore, the only limits on how high you can set the velocity feedforward gain (SGVF) are: when it *saturates the control output* (tries to exceed the servo

controller's ±10V analog control signal range); or when it causes the actual position to *precede* the commanded position.

---

**NOTE**

The SGVF command is specific to the current feedback source (selected with the last SFB command). Therefore, if your application requires switching between feedback sources on the same axis, then for each feedback source, you must select the feedback source with the appropriate SFB command and then issue the SGVF command with the gain values specific to the selected feedback source.

---

For more information on servo tuning, refer to your product's *Installation Guide* or to the Motion Planner help system.

**Example:**
```
SGVF3555,3555,4000,4000   ; Sets the velocity feedforward for all axes
```

---

# [ SIN( ) ]    Sine

| | | | |
|---|---|---|---|
| Type | Operator (Trigonometric) | **Product** | **Rev** |
| Syntax | ... SIN(r) (See below) | 6K | 5.0 |
| Units | r = value in radian or degrees based on RADIAN command | | |
| Range | ±17500.0000000 radians | | |
| Default | n/a | | |
| Response | n/a | | |
| See Also | [ ATAN ], [ COS ], [ PI ], RADIAN, [ TAN ], VAR | | |

This operator is used to calculate the sine of a number given in radians or degrees (see the RADIAN command). If "a" and "b" are coordinates of a point on a circle of radius "r", then the angle of measure "θ" can be defined by the equation: $\sin \theta = \dfrac{a}{r}$
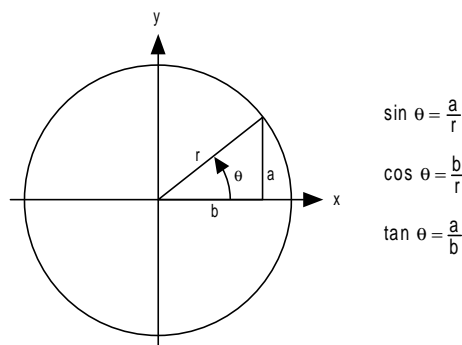
If a value is given in radians and a conversion is needed to degrees, use the formula: $360° = 2\pi$ radians.



$$\sin \theta = \frac{a}{r}$$
$$\cos \theta = \frac{b}{r}$$
$$\tan \theta = \frac{a}{b}$$

The graph on the right shows the amplitude of **y** on the unit circle for different values of **x**.
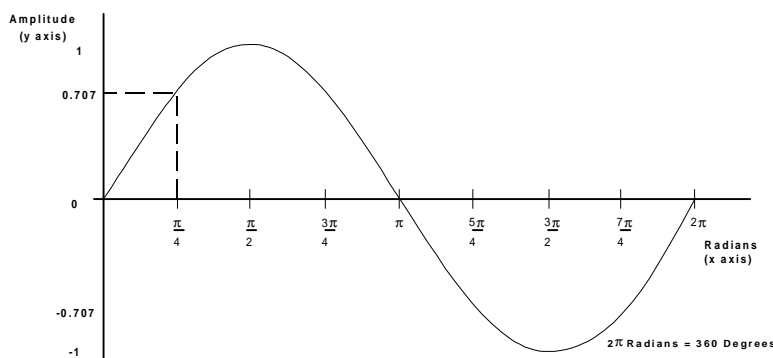


**Syntax:** VARx=SIN(r) where "x" is the numeric variable number and "r" is a value provided in either degrees or radians based on the RADIAN command. Parentheses ( () ) must be placed around the SIN operand. The result will be specified to 5 decimal places.

**Example:**
```
RADIAN1
VAR1=5 * SIN(PI/4) ; Set variable 1 equal to 5 times the sine of Pi divided by 4
```

# SINAMP          Virtual Master Sine Wave Amplitude

| | | | **Product** | **Rev** |
|---|---|---|---|---|
| Type | `Following` | | 6K | 5.0 |
| Syntax | `<!><@><a>SINAMP<i>,<i>,<i>,<i>,<i>,<i>,<i>,<i>` | | | |
| Units | `i = amplitude` | | | |
| Range | `0-8192 (max. peak to peak is 16384)` | | | |
| Default | `0` | | | |
| Response | `SINAMP   *SINAMP0,0,0,0,0,0,0,0` | | | |
| | `1SINAMP  *1SINAMP0` | | | |
| See Also | `FOLMAS, FVMACC, FVMFRQ, SINANG, SINGO` | | | |

Use the `SINAMP` command to define the amplitude of the internal sine wave when it has been designated as the virtual master.  By designating the internal sine wave as a master, the user may produce a sinusoidally oscillating motion, with control of the phase, amplitude, and center of oscillation.

The `SINAMP` command allows a change in follower amplitude without changing the center of oscillation.  It affects the sine wave immediately, without any built in ramp in amplitude.  If a gentle change in amplitude is desired, write a user program which repeatedly issues the command with small changes in value until the desired value is reached.

The peak-to-peak amplitude of a virtual master sine wave is twice the value specified with the `SINAMP` command.


# SINANG          Virtual Master Sine Wave Angle

| | | | **Product** | **Rev** |
|---|---|---|---|---|
| Type | `Following` | | 6K | 5.0 |
| Syntax | `<!><@><a>SINANG<r>,<r>,<r>,<r>,<r>,<r>,<r>,<r>` | | | |
| Units | `degrees` | | | |
| Range | `0.0-360.0` | | | |
| Default | `0` | | | |
| Response | `SINAMP   *SINAMP0,0,0,0,0,0,0,0` | | | |
| | `1SINAMP  *1SINAMP0` | | | |
| See Also | `FOLMAS, FVMACC, FVMFRQ, SINAMP, SINGO` | | | |

The `SINANG` command is used to define the phase angle when the internal sine wave is designated as the virtual master.  By designating the internal sine wave as a master,  the user may produce a sinusoidally oscillating motion,  with control of the phase, amplitude, and center of oscillation.

There is one sine wave per axis, each using the variable count frequency (`FVMFRQ`) of that axis to increase or decrease the angle from which the sine is calculated.  Each count of the count frequency  changes the angle by one-tenth (0.1) of a degree.  For example, a `FVMFRQ` value of 3600 would create an angular frequency of 3600 tenths of degrees per second, or 1 cycle per second.  When used as a source for the sine wave, the maximum value for `FVMFRQ` is 144000.  This results in a maximum of 40 Hz angular frequency. Frequencies higher than this are not allowed because they may be subject to aliasing.


# SINGO          Virtual Master - Initiate Internal Sine Wave

| | | | **Product** | **Rev** |
|---|---|---|---|---|
| Type | `Following` | | 6K | 5.0 |
| Syntax | `<!><@><a>SINGO<b><b><b><b><b><b><b><b>` | | | |
| Units | `n/a` | | | |
| Range | `b = 1 (restart sine wave from previous angle & amplitude) or` | | | |
| | `0 (stop sine wave)` | | | |
| Default | `0` | | | |
| Response | `SINGO    *SINGO0000_0000` | | | |
| | `1SINGO   *1SINGO0` | | | |
| See Also | `FOLMAS, FVMACC, FVMFRQ, SINAMP, SINANG` | | | |

The `SINGO` command is used to restart the internal sine wave from zero degrees.  By designating the internal sine wave as a master,  the user may produce a sinusoidally oscillating motion, with control of the phase, amplitude, and center of oscillation.

The SINGO command with a "0" parameter abruptly stops the sine wave, without changing its current magnitude. Using the SINGO command with a "1" parameter abruptly starts the sine wave, also without changing its current magnitude. To gently pause the follower output, change the FVMFRQ value to zero with a moderate FVMACC value; to resume the follower output, restore the original FVMFRQ value.

The SINGO command with a "1" parameter always starts at the previous angle, which may not be the desired start of oscillation. The SINANG command will instantly change the angle and corresponding sine of the angle. This represents an abrupt change in master position. If the follower axis is still following when this occurs, there will be an abrupt change in commanded follower position. To start the follower properly, move the follower to the desired start position first (using MC0, D, GO), then issue SINANG, then MC1, GO1, and finally SINGO.

---

## SMPER      Maximum Allowable Position Error

| | | | |
|---|---|---|---|
| Type | Servo | **Product** | **Rev** |
| Syntax | <!><@><a>SMPER<r>,<r>,<r>,<r>,<r>,<r>,<r>,<r> | 6K | 5.0 |
| Units | r = feedback device steps (scalable with SCLD) | | |
| Range | 0-200,000,000 (0 = do not monitor position error condition) | (applicable to | |
| Default | 4000 | servo axes only) | |
| Response | SMPER:   *SMPER4000,4000,4000,4000 ... | | |
| | 1SMPER:  *1SMPER4000 | | |
| See Also | [ AS ], CMDDIR, ENCPOL, [ ER ], ERES, ERROR, ERRORP, SCALE, SCLD, SFB, SGILIM, TANI, TAS, TER, TFB, TPC, TPE, TPER | | |

This command allows you to set the maximum position error allowed before an error condition occurs. The position error, monitored once per system update period, is the difference between the commanded position and the actual position as read by the feedback device selected with the last SFB command. When the position error exceeds the value entered by the SMPER command, an error condition is latched (see TAS or AS bit #23) and the 6K controller issues a shutdown to the faulted axis and sets its analog output command to zero volts. To enable the system again, the DRIVE1 command must be issued to the affected axis, which also sets the commanded position equal to the actual feedback device position (incremental devices will be zeroed).

If the SMPER value is set to zero (SMPER∅), the position error condition is not monitored, allowing the position error to accumulate without causing a fault.

When SMPER is set to a non-zero value, the maximum position error acts as the servo system fault monitor; if the system becomes unstable or loses position feedback, the controller detects the resulting position error, shuts down the drive, and sets an error status bit. You can enable ERROR command bit #12 to continually check for the position error condition, and when it occurs to branch to a programmed response defined in the ERRORP program. You can check the status of this error condition with the TAS, AS, TER, and ER commands. You can check the actual position error with the TPER and PER commands.

If scaling is enabled (SCALE1), the SMPER value is multiplied by the SCLD value.

---

**NOTE**

The SMPER command is specific to the current feedback source (selected with the last SFB command). Therefore, if your application requires switching between feedback sources on the same axis, then for each feedback source, you must select the feedback source with the appropriate SFB command and then issue the SMPER command with the gain values specific to the selected feedback source.

---

**Example:**
```
ERES4000,4000,4000,4000   ; Set encoder resolution for all axes to 4000 counts/rev
SMPER4000,4000,4000,4000   ; Set maximum allowable position error to 1 rev for
                           ; all 4 axes. If the position error exceeds 4000 counts
                           ; (1 rev) a fault condition will occur.
```

# SOFFS          Servo Control Signal Offset

| Type | Servo | | **Product** | **Rev** |
|------|-------|---|---------|-----|
| Syntax | <!><@><a>SOFFS<r>,<r>,<r>,<r>,<r>,<r>,<r>,<r> | | 6K | 5.0 |
| Units | r = volts | | | |
| Range | -10.000 to 10.000 (resolution is 0.001 volts) | | | |
| Default | 0 | | | |
| Response | SOFFS:   *SOFFS0,0,0,0,0,0,0,0 | | | |
| | 1SOFFS:  *1SOFFS0 | | | |
| See Also | [ DAC ], DACLIM, SGENB, SGSET, TDAC, TGAIN, TSGSET | | | |

This command allows you to set an offset voltage to the commanded analog control signal output (commanded analog output + SOFFS value = offset analog output). With this command, you can set an offset voltage to the drive system so that the motor will be stationary in an open-loop configuration. *This is the same effect as the balance input on most analog servo drives.*

---

**CAUTION**

If there is little or no load attached, the SOFFS offset may cause an acceleration to a high speed.

---

Typically, this offset will be set to zero. This offers a method for setting the analog output command to a known voltage. By setting the SGP, SGI, SGV, SGAF, & SGVF gains to zero, the analog output will reflect this offset value and the system becomes an open-loop configuration.

Use the TDAC command to check the voltage being commanded at the 6K controller's analog output (voltage displayed includes any offset in effect). An axis configured as a stepper <u>can</u> use the SOFFS command to set the DAC output voltage.

**Example:**
```
SOFFS0,0,1,2   ; Sets the offset voltage on all axes
```

# [ SQRT() ]    Square Root

| Type | Operator (Mathematical) | | **Product** | **Rev** |
|------|------------------------|---|---------|-----|
| Syntax | See below | | 6K | 5.0 |
| Units | n/a | | | |
| Range | n/a | | | |
| Default | n/a | | | |
| Response | n/a | | | |
| See Also | [ = ], [ + ], [ – ], [ * ],[ / ], VAR | | | |

This operator takes the square root of a value. The result, if multiplied by itself, will *approximately equal* the original value (the difference is attributed to round-off error). The resulting value has 3 decimal places.

Syntax: VARn=SQRT(expression) where "n" is the variable number, and the expression can be a number or a mathematical expression. The SQRT of a negative number is not allowed. Parentheses ( () ) must be placed around the SQRT operand.

**Example:**
```
VAR1=SQRT(25)   ; Set variable 1 equal to the square root of 25 (result = 5)
```

# [ SS ]          System 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 | IF, TCMDER, TRGFN, [ TRIG ], TTRIG, TSS, TSSF, TSTAT, VARB | | | |

Use the SS operator to assign the system status bits to a binary variable (VARB), 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, Ø, 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 Ø through 9.

**Syntax:** `VARBn=<i%>SS` where "n" is the binary variable number, or `SS` can be used in an expression such as `IF(SS=b11Ø1)`, or `IF(SS=h7F)`. **NOTE**: If you are using <u>multi-tasking</u>, be aware that each task has its own system status register. If you wish to check the system status of a external task (a task other than the task that is executing the `SS` operator), then you must prefix the `SS` operator to address the targeted task (e.g., `2%SS` for the system status of Task 2).

The function of each system status bit is shown below.

| **BIT** (Left to Right) | **Function** (1 = yes, ∅ = no) | **BIT** (Left to Right) | **Function** (1 = yes, ∅ = no) |
|---|---|---|---|
| 1 | System Ready | 17 | Loading Thumbwheel Data (`TW`) |
| 2 | Reserved | 18 | External Program Select Mode (`INSELP`) |
| 3 | Executing a Program | 19 | Dwell in Progress (`T` command) |
| 4 | Immediate Command (set if last command was immediate) | 20 | Waiting for RP240 Data—`DREAD` or `DREADF` |
| 5 | In ASCII Mode | 21 | RP240 Connected — current `PORT` setting only |
| 6 | In Echo Mode — current `PORT` setting only | 22 | Non-volatile Memory Error |
| 7 | Defining a Program | 23 | Servo data gathering transmission in progress (servo axes only) |
| 8 | In Trace Mode | 24 | Reserved |
| 9 | In Step Mode | 25 | RESERVED |
| 10 | In Translation Mode | 26 | RESERVED |
| 11 | Command Error Occurred (bit is cleared when `TCMDER` is issued) | 26 | RESERVED |
| 12 | Break Point Active (`BP`) | 28 | RESERVED |
| 13 | Pause Active | 29 | Compiled memory is 75% full |
| 14 | Wait Active (`WAIT`) | 30 | Compiled memory is 100% full |
| 15 | Monitoring On Condition (`ONCOND`) | 31 * | Compile operation failed (`PCOMP`) ** |
| 16 | Waiting for Data (`READ`) | 32 | Reserved |

\* Bit #31: failed `PCOMP` compile is cleared on power up, `RESET`, or after successful compile. Possible causes include:
- Errors in profile design (e.g., change direction while at non-zero velocity; distance & velocity equate to < 1 count per system update; preset move profile ends in non-zero velocity)
- Profile will cause a Following error (see `TFSF`, `TFS`, or `FS` command descriptions)
- Out of memory (see `SS` bit #30)
- Axis already in motion at the time of the `PCOMP` command
- Loop programming errors (e.g., no matching `PLOOP` or `PLN`; more than 4 embedded `PLOOP`/`END` loops)
- `PLCP` program contains invalid commands.

If it is desired to assign only one bit of the system status value to a binary variable, instead of all 32, the bit select (.) operator can be used. For example, `VARB1=SS.12` assigns system status bit 12 to binary variable 1: `*VARB1=XXXX_XXXX_XXXØ_XXXX_XXXX_XXXX_XXXX_XXXX`.

**Example:**
```
VARB1=SS   ; System status assigned to binary variable 1
IF(SS=b111011X11) ; If the system status contains 1s in bit locations 1, 2, 3,
                  ; 5, 6, 8, & 9, and a 0 in bit location 4, do the IF
                  ; statement
IF(SS=h7F00)   ; If the system status contains 1s in bit locations 1, 2, 3,
                  ; 5, 6, 7, & 8, and 0s in every other bit location, do the IF
                  ; statement
NIF   ; End of second IF statement
NIF   ; End of first IF statement
```

# STARTP    Start-Up Program

| | | | Product | Rev |
|---|---|---|---|---|
| Type | Subroutines | | 6K | 5.0 |
| Syntax | <!>STARTP<t> | | | |
| Units | t = text (name of program) | | | |
| Range | Text name of 6 characters or less | | | |
| Default | n/a | | | |
| Response | STARTP: *STARTP MAIN | | | |
| See Also | DEF, RESET, SCALE | | | |

The Start-Up Program (STARTP) command specifies the name of the program that will automatically when the 6K product is powered up or reset with the RESET command. If the program that is identified as the STARTP program is deleted with the DEL command, the STARTP is automatically cleared. If you wish to prevent the STARTP program from being executed, without having to delete the assigned program, issue the STARTP CLR command.

**Example:**
```
STARTP WakeUp   ; Set program WakeUp as the program that will start to run
                ; after power is cycled or the 6K product is reset
STARTP CLR   ; Clears the program WakeUp from its assignment as the
                ; start-up program
DEL WakeUp   ; Deletes the program WakeUp and clears the STARTP command
                ; (no power-up program will be executed)
```

# STEP    Single Step Mode Enable

| | | | Product | Rev |
|---|---|---|---|---|
| Type | Program Debug Tool | | 6K | 5.0 |
| Syntax | <!>STEP<b> | | | |
| Units | n/a | | | |
| Range | b = 0 (disable), 1 (enable) or X (don't care) | | | |
| Default | 0 | | | |
| Response | STEP:    *STEP0 | | | |
| See Also | [ # ], BP, [ SS ], TRACE, TRACEP, TRANS, TSS | | | |

The Single Step Mode Enable (STEP) command enables single command step mode. Single step mode is used for stepping through a defined (DEF) program. To execute single step mode:

1. Define a program (DEF)
2. Enable single step mode (STEP1)
3. Run the program (RUN)
4. Use the immediate pound (!#) to step through the program

Each step (!#) command will initiate the next command to be processed.

**Example:**
```
DEF tester   ; Begin definition of program named tester
V1,1,1,1   ; Set velocity to 1 unit/sec on all axes
A10,10,10,10   ; Set acceleration to 10 units/sec/sec on all axes
                ; (Note: This command will not be executed until a !# sign
                ; is received.)
D1,2,3,4   ; Set distance to 1 unit on axis 1, 2 units on axis 2,
                ; 3 units on axis 3, and 4 units on axis 4
GO1101   ; Initiate motion on axes 1, 2, and 4
OUT11X1   ; Turn on onboard outputs 1, 2, and 4, leave 3 unchanged
END   ; End program definition
STEP1   ; Enable single step mode
RUN tester   ; Execute program named tester
; ***
; * At this point no action will occur because single step mode   *
; * has been enabled. Here's how to execute commands: *
; * !#2 (Execute 1st 2 commands in the program: V1,1,1,1 & A10,10,10,10) *
; * !# (Execute 1 command: command to be executed is D1,2,3,4)   *
; * !#1 (Execute 1 command: command to be executed is GO110)   *
; * !#2 (Execute 2 commands: commands to be executed are OUT11X1 & END)   *
; ***
```

# STRGTD     Target Distance Zone

| | | | |
|---|---|---|---|
| Type | Servo | **Product** | **Rev** |
| Syntax | `<!><@><a>STRGTD<r>,<r>,<r>,<r>,<r>,<r>,<r>,<r>` | 6K | 5.0 |
| Units | `r = distance units (scalable with SCLD)` | | |
| Range | `0-999,999,999.99999` | | |
| Default | `50` | (applicable only to servo axes) | |
| Response | `STRGTD:  *STRGTD50,50,50,50 ...`<br>`1STRGTD: *1STRGTD50` | | |
| See Also | `[ AS ], SCLD, STRGTE, STRGTT, STRGTV, TAS, TSTLT` | | |

This command sets the target distance zone used in the Target Zone Settling Mode. The target distance zone is a range of positions around the desired endpoint that the load must be within before motion is considered complete. If scaling is enabled (SCALE1), the STRGTD value is multiplied by the distance scale factor (SCLD).

When using the Target Zone Mode, the load's actual position and actual velocity must be within the *target zone* (that is, within the distance zone defined by STRGTD and within the velocity zone defined by STRGTV) before motion can be determined complete. Axis status bit #24 (see TASF, TAS, or AS) indicates when the axis is within the zone specified with STRGTD and STRGTV; this bit is usable even if the Target Zone Mode is not enabled (STRGTE0).

If the load does not settle into the target zone before the timeout period set by STRGTT, the controller detects an error (see TASF, TAS, or AS bit #25). If this error occurs, you can prevent subsequent command and/or move execution by enabling the ERROR command to continually check for this error condition, and when it occurs to branch to a programmed response defined in the ERRORP program. (Refer to the ERRORP command description for an example of using an error program.)

*\*\*\* For a more information on target zone operation, refer to the Programmer's Guide.*

**Example:**
```
STRGTD5,5,5,5   ; Sets the distance target zone to +/-5 units
STRGTV.01,.01,.01,.01 ; Sets the velocity target zone to <= 0.01 units/sec
STRGTT10,10,10,10   ; Sets the timeout period to 10 milliseconds on all axes
STRGTE1111   ; Enables the target zone criterion for all axes
;
; Given these target zone commands, a move with a distance of 8,000 units
; (@D8000) must end up between position 7,995 and 8,005 and settle down
; to <=0.01 units/sec within 10 ms after the commanded profile is complete.
```

# STRGTE     Enable Target Zone Settling Mode

| | | | |
|---|---|---|---|
| Type | Servo | **Product** | **Rev** |
| Syntax | `<!><@><a>STRGTE<b><b><b><b><b><b><b><b>` | 6K | 5.0 |
| Units | n/a | | |
| Range | `b = 0 (disable), 1 (enable), or X (don't care)` | | |
| Default | `0` | (applicable only to servo axes) | |
| Response | `STRGTE:  *STRGTE0000_0000`<br>`1STRGTE:  *1STRGTE0` | | |
| See Also | `COMEXC, STRGTD, STRGTT, STRGTV, TSTLT` | | |

This command enables or disables the Target Zone Settling Mode. When using the target zone settling criterion, the load's actual position and actual velocity must be within the *target zone* (that is, within the position band defined by STRGTD and within the velocity band defined by STRGTV) before motion can be determined complete.

If the load does not settle into the target zone before the timeout period set by STRGTT, the controller detects an error (see TAS or AS bit #25). If this error occurs, you can prevent subsequent command and/or move execution by enabling the ERROR command to continually check for this error condition, and when it occurs to branch to a programmed response defined in the ERRORP program.

*\*\*\* For a more information on target zone operation, refer to the Programmer's Guide.*

**Example:**
```
STRGTD5,5,5,5   ; Sets the distance target zone to +/-5 units
STRGTV.01,.01,.01,.01 ; Sets the velocity target zone to <= 0.01 units/sec
STRGTT10,10,10,10   ; Sets the timeout period to 10 milliseconds on all axes
STRGTE1111   ; Enables the target zone criterion for all axes
;
; Given these target zone commands, a move with a distance of 8,000 units
; (@D8000) must end up between position 7,995 and 8,005 and settle down
; to <=0.01 units/sec within 10 ms after the commanded profile is complete.
```

---

# STRGTT   Target Settling Timeout Period

| | | | |
|---|---|---|---|
| Type | Servo | **Product** | **Rev** |
| Syntax | `<!><@><a>STRGTT<i>,<i>,<i>,<i>,<i>,<i>,<i>,<i>` | 6K | 5.0 |
| Units | r = milliseconds | | |
| Range | 0-5000 | (applicable only to servo axes) | |
| Default | 1000 | | |
| Response | STRGTT:   *STRGTT1000,1000,1000,1000 ... | | |
| | 1STRGTT:  *1STRGTT1000 | | |
| See Also | [ AS ], [ ER ], ERROR, ERRORP, STRGTD, STRGTE, STRGTV, TAS, TER, TSTLT | | |

This command sets the maximum time allowed for the load to settle within the defined target zone before an error occurs.

This command is useful only if the Target Zone Settling Mode is enabled with the STRGTE command. When using the Target Zone Settling Mode, the load's actual position and actual velocity must be within the *target zone* (that is, within the position band defined by STRGTD and within the velocity zone defined by STRGTV) before motion can be determined complete. If the load does not settle into the target zone before the timeout period set by STRGTT, the servo controller detects an error (see TAS or AS bit #25).

If this error occurs, you can prevent subsequent command and/or move execution by enabling the ERROR command to continually check for this error condition, and when it occurs to branch to a programmed response defined in the ERRORP program. (Refer to the ERRORP command description for an example of using an error program.) You can check the status of the error condition with the TER and ER commands.

*\*\*\* For a more information on target zone operation, refer to the Programmer's Guide.*

**Example** (see STRGTE):

---

# STRGTV   Target Velocity Zone

| | | | |
|---|---|---|---|
| Type | Servo | **Product** | **Rev** |
| Syntax | `<!><@><a>STRGTV<r>,<r>,<r>,<r>,<r>,<r>,<r>,<r>` | 6K | 5.0 |
| Units | r = units/sec (scalable by SCLV) | | |
| Range | 0.00000-1,600,000.00000 | (applicable only to servo axes) | |
| Default | 1.00000 | | |
| Response | STRGTV:   *STRGTV1.0000,1.0000,1.0000,1.0000 ... | | |
| | 1STRGTV:  *1STRGTV1.0000 | | |
| See Also | [ AS ], SCLV, STRGTD, STRGTE, STRGTT, TAS, TSTLT | | |

This command sets the target velocity zone for use in the Target Zone Settling Mode. The target velocity zone is a velocity range that the load must be within before motion is considered complete. If scaling (SCALE) is enabled, the STRGTV value is multiplied by the velocity scale factor (SCLV).

When using the Target Zone Mode, the load's actual position and actual velocity must be within the *target zone* (that is, within the distance zone defined by STRGTD and less than or equal to the velocity defined by STRGTV) before motion can be determined complete. Axis status bit #24 (see TASF, TAS, or AS) indicates when the axis is within the zone specified with STRGTD and STRGTV; this bit is usable even if the Target Zone Mode is not enabled (STRGTE0).

If the load does not settle into the target zone before the timeout period set by STRGTT, the servo controller detects an error (see TAS or AS bit #25). If this error occurs, you can prevent subsequent command and/or move execution by enabling the ERROR command to continually check for this error condition, and when it occurs to branch to a programmed response defined in the ERRORP program. (Refer to the ERRORP command description for an example of using an error program.)

*** *For a more information on target zone operation, refer to the Programmer's Guide.*

**Example:**
```
STRGTD5,5,5,5   ; Sets the distance target zone to +/-5 units
STRGTV.01,.01,.01,.01 ; Sets the velocity target zone to <= 0.01 units/sec
STRGTT10,10,10,10   ; Sets the timeout period to 10 milliseconds on all axes
STRGTE1111   ; Enables the target zone criterion for all axes
;
; Given these target zone commands, a move with a distance of 8,000 units
; (@D8000) must end up between position 7,995 and 8,005 and settle down
; to <=0.01 units/sec within 10 ms after the commanded profile is complete.
```

---

## [ SWAP ]    Task Swap Assignment

| 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 | %, [ SS ], TTASK, TSWAP, TSKTRN, TSKAX, TSS | | |

The Task Swap Assignment command (SWAP) allows a binary bit pattern indicating the tasks that are currently active to be assigned to a binary variable, or evaluated in a conditional statement such as IF or WAIT. This is useful for ascertaining which tasks have any activity. To ascertain exactly what activity a specific task has at a given time, use the system status (SS or TSS).

SWAP's binary 10-bit pattern represents tasks 1-10, from left to right. A "1" indicates that the task is active, and a "0" indicates that the task is inactive. The "Task Supervisor", represented by task Ø, is always active and is therefore not included in the SWAP and TSWAP status.

**Syntax**: VARBn=SWAP where "n" is the binary variable number, or SWAP can be used in an expression such as IF(SWAP=b1001000000) or IF(SWAP.3=b1) or IF(SWAP=h7F0).

To check the status of only one task, you may use the bit select (.) operator. For example, VARB1=SWAP.2 assigns the binary state of Task2 to binary variable 1; or WAIT(SWAP.2=b1) establishes a wait condition that evaluates true when Task2 becomes active.

# T — Time Delay

| | | | |
|---|---|---|---|
| Type | `Program Flow Control` | **Product** | **Rev** |
| Syntax | `<!>T<r>` | 6K | 5.0 |
| Units | `r = seconds` | | |
| Range | `0.001-999.999` | | |
| Default | `n/a` | | |
| Response | `n/a` | | |
| See Also | `GOWHEN, PS, [ SS ], [ TIM ], TTIM, TSS, WAIT` | | |

The Time Delay (`T`) command pauses command processing for **r** seconds before continuing command execution. Once the elapsed time has expired, the command after the `T` command will be executed.

The minimum resolution of the `T` command is 2 ms. Although you can enter time delays that are not multiples of 2 ms, the time delay will be rounded up to the next multiple of 2 ms. For example, `T.005` produces a 6 ms time delay.

**Example:**
```
T5   ; Wait 5 seconds before executing TPE command
TPE   ; Transfer position of all encoders to the terminal
```

# [ TAN( ) ] — Tangent

| | | | |
|---|---|---|---|
| Type | `Operator (Trigonometric)` | **Product** | **Rev** |
| Syntax | `... TAN(r) (See below)` | 6K | 5.0 |
| Units | `r = radians or degrees depending on RADIAN command` | | |
| Range | `±17500.0000000 radians` | | |
| Default | `n/a` | | |
| Response | `n/a` | | |
| See Also | `[ ATAN ], [ COS ], [ PI ], RADIAN, [ SIN ], VAR` | | |

The Tangent (`TAN`) operator is used to calculate the tangent of a number given in radians or degrees (see the `RADIAN` command). If "a" and "b" are coordinates of a point on a circle of radius "r", then the angle of measure "θ" can be defined by the equation: $\tan \theta = \dfrac{a}{b}$

If a value is given in radians and a conversion is needed to degrees, use the following formula: $360° = 2\pi$ radians.

Syntax: `VARx=TAN(r)`, where `x` is the numeric variable number and `r` is a value in either radians or degrees depending on the `RADIAN` command. Parentheses ( `( )` ) must be placed around the `TAN` operand. The result will be specified to 5 decimal places.

$$\sin \theta = \frac{a}{r}$$

$$\cos \theta = \frac{b}{r}$$

$$\tan \theta = \frac{a}{b}$$

**Example:**
```
VAR1=5 * TAN(PI/4)   ; Set variable 1 = 5 times the tangent of Pi divided by 4
```
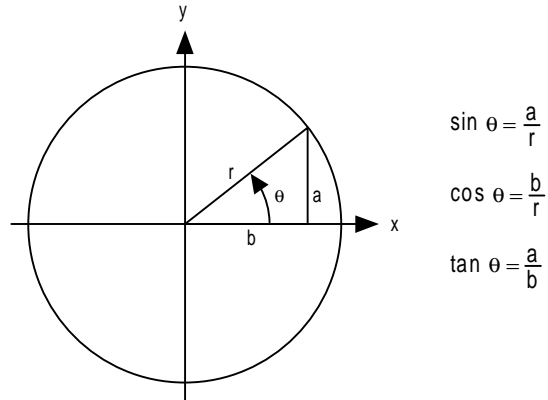
## TANI — Transfer Analog Input Voltage

| | | | |
|---|---|---|---|
| **Type** | Transfer | **Product** | **Rev** |
| Syntax | <!><B>TANI<.i> | 6K | 5.0 |
| Units | B = I/O brick | | |
| | i = location on I/O brick | | |
| Range | i = 1-32 | | |
| Default | n/a | | |
| Response | 1TANI    *1TANIx,x,x,x,x,x,x,x | | |
| | +5.802,-4.663,-4.972, +6.023,+2.126,+2.223, ... | | |
| | x,x,x,x,x,x,x,x | | |
| | x,x,x,x,x,x,x,x | | |
| | 1TANI.10 *-4.663 | | |
| See Also | ANIRNG, [ ANI ], [ FB ], [ PANI ], TFB, TPANI | | |

The Transfer Analog Input Voltage for analog inputs (TANI) command returns the voltage level present at the ANI analog inputs located on external I/O bricks. The value reported with the TANI command is measured in volts and does not reflect the effects of distance scaling (SCLD), position offset (PSET), or commanded direction polarity (CMDDIR). To ascertain the offset ANI input value, as affected by SCLD, PSET, or CMDDIR, use the TPANI command or the TFB command.

To determine the analog value from a specific input, use the bit select operator (.). For example, to check the voltage of the 2$^{nd}$ analog input on the 3$^{rd}$ SIM (I/O location 18) of I/O brick 2, use the 2TANI.18 command. To understand more about the location of I/O points on external I/O bricks, see page 6.

The TANI value is derived from the voltage applied to the corresponding analog input and ground. The analog value is determined from a 12-bit analog-to-digital converter (ADC). Under the default ANI voltage range, set with ANIRNG, the range of the ANI operator is -10.000VDC to +10.000VDC (see ANIRNG command for optional voltage ranges).

## TAS — Transfer Axis Status

| | | | |
|---|---|---|---|
| **Type** | Transfer | **Product** | **Rev** |
| Syntax | <!><a>TAS<.i> | 6K | 5.0 |
| Units | i = bit location on the specified axis (See below) | | |
| Range | 1-32 | | |
| Default | n/a | | |
| Response | TAS:    *TAS 0000_0000_0000_0000_0000_0000_0000_0000 | | |
| | *    0000_1000_0000_0000_0000_0000_0000_0000 | | |
| | *    (repeated for each axis) | | |
| | 1TAS:   *1TAS0000_0000_0000_0000_0000_0000_0000_0000 | | |
| | bit 1 ⟶                                              ⟵ bit 32 | | |
| | TAS.5:   *00110000 (bit 5 of all eight axes status registers) | | |
| | 1TAS.5:  *0 (bit 5 of status register for axis 1) | | |
| See Also | [ AS ], [ ASX ], DRFLVL, ESTALL, GOWHEN, HOM, JOG, JOY, MA, MC, SMPER, STRGTD, STRGTE, STRGTT, STRGTV, TASF, TASX, TSTAT | | |

The Transfer Axis Status (TAS) command returns the current status of all axes.

---

**FULL-TEXT STATUS REPORT AVAILABLE**

The TAS status command reports a binary bit report.  If you would like to see a more descriptive text-based report, use the TASF command description.

---

| Bit #<br>(left to right) | Function (1/∅) |
|---|---|
| 1 | Moving/Not Moving. This bit is set only when motion is <u>commanded</u> on the axis. The motor may still be "moving" (e.g., due to end-of-move settling). |
| 2 | Negative/positive-direction |
| 3 | Accelerating/Not Accelerating. This bit does not indicate deceleration (bit is set to 0 during decel); to check if the axis is decelerating, the state of TAS bits 1, 3 and 4 should be: TAS1x00. |
| 4 | At Velocity/Not at Velocity |
| 5 | Home Successful (HOM) (YES/NO) |
| 6 | Absolute/Incremental (MA) |
| 7 | Continuous/Preset (MC) |
| 8 | Jog Mode/Not Jog Mode (JOG) |
| 9 | Joystick Mode/Not Joystick Mode (JOY) |
| 10 | RESERVED |
| 11 | RESERVED |
| 12 | Stall Detected (YES/NO). This bit is not usable until Stall Detect is enabled with ESTALL1 command. |
| 13 | Drive Shut Down (YES/NO) |
| 14 | Drive Fault occurred (YES/NO). A drive fault cannot be detected (this bit is always 0) until the drive fault input check is enabled with DRFEN1. <u>Note</u>: TASX bit 4 reports the hardware state of the drive fault input, regardless of DRFEN or DRIVE. |
| 15 | Positive-direction Hardware Limit Hit (YES/NO) |
| 16 | Negative-direction Hardware Limit Hit (YES/NO) |
| 17 | Positive-direction Software Limit Hit (YES/NO) |
| 18 | Negative-direction Software Limit Hit (YES/NO) |
| 19 | RESERVED |
| 20 | RESERVED |
| 21 | RESERVED |
| 22 | RESERVED |
| 23 | Position Error Exceeded (SMPER) (YES/NO). Servo axes only. |
| 24 | In Target Zone (defined with STRGTD & STRGTV) (YES/NO). Servo axes only. This bit is set only after the *successful completion* of a move (if STRGTD and STRGTV criteria have been satisfied). This bit is usable even if the Target Zone mode is not enabled (STRGTE0). |
| 25 | Target Zone Timeout occurred (STRGTT) (YES/NO). Servo axes only. |
| 26 | Change in motion is suspended pending GOWHEN (YES/NO). This bit is cleared if the GOWHEN condition is true, or if STOP (!S) or KILL (!K or ^K) is executed. |
| 27 | RESERVED |
| 28 | Registration move initiated by trigger since last GO command. This bit is cleared with the next GO command. |
| 29 | RESERVED |
| 30 | Pre-emptive (OTF) GO or Registration profile not possible |
| 31 | RESERVED |
| 32 | RESERVED |

# TASF          Transfer Axis Status (full-text report)

| | | **Product** | **Rev** |
|---|---|---|---|
| Type | `Transfer` | 6K | 5.0 |
| Syntax | `<!><a>TASF` | | |
| Units | `n/a` | | |
| Range | `n/a` | | |
| Default | `n/a` | | |
| Response | `TASF:    (see example below)` | | |
| See Also | `[ AS ], [ ASX ], DRFLVL, ESTALL, GOWHEN, HOM, JOG, JOY, MA,` | | |
| | `MC, SMPER, STRGTD, STRGTE, STRGTT, STRGTV, TAS, TASX, TSTAT` | | |

The `TASF` command returns a text-based status report of all axes.  This is an alternative to the binary report (`TAS`).  Example `TASF` response:

```
*TASF               AXIS #
*                   1    2    3    4    5    6    7    8
*Moving             NO   NO   NO   NO   NO   NO   NO   NO
*Direction NEG      NO   NO   NO   NO   NO   NO   NO   NO
*Accelerating       NO   NO   NO   NO   NO   NO   NO   NO
*At Velocity        NO   NO   NO   NO   NO   NO   NO   NO
*
*Home successful    NO   NO   NO   NO   NO   NO   NO   NO
*Mode Absolute      NO   NO   NO   NO   NO   NO   NO   NO
*Mode Continuous    NO   NO   NO   NO   NO   NO   NO   NO
*Jog Mode           NO   NO   NO   NO   NO   NO   NO   NO
*
*Joystick Mode      NO   NO   NO   NO   NO   NO   NO   NO
*RESERVED           NO   NO   NO   NO   NO   NO   NO   NO
*RESERVED           NO   NO   NO   NO   NO   NO   NO   NO
*Stall Detected     NO   NO   NO   NO   NO   NO   NO   NO
*
*Drive Shutdown     NO   NO   NO   NO   NO   NO   NO   NO
*Drive Faulted      NO   NO   NO   NO   NO   NO   NO   NO
*POS Hard Limit Hit NO   NO   NO   NO   NO   NO   NO   NO
*NEG Hard Limit Hit NO   NO   NO   NO   NO   NO   NO   NO
*
*POS Sftwr Limit Hit NO   NO   NO   NO   NO   NO   NO   NO
*NEG Sftwr Limit Hit NO   NO   NO   NO   NO   NO   NO   NO
*RESERVED           NO   NO   NO   NO   NO   NO   NO   NO
*RESERVED           NO   NO   NO   NO   NO   NO   NO   NO
*
*RESERVED           NO   NO   NO   NO   NO   NO   NO   NO
*RESERVED           NO   NO   NO   NO   NO   NO   NO   NO
*Pos Error Exceeded NO   NO   NO   NO   NO   NO   NO   NO
*In Target Zone     YES  YES  YES  YES  YES  YES  YES  YES
*
*Target Zone Timeout NO   NO   NO   NO   NO   NO   NO   NO
*Gowhen is Pending  NO   NO   NO   NO   NO   NO   NO   NO
*RESERVED           NO   NO   NO   NO   NO   NO   NO   NO
*Reg Move Commanded NO   NO   NO   NO   NO   NO   NO   NO
*
*RESERVED           NO   NO   NO   NO   NO   NO   NO   NO
*Preset Move OvershotNO  NO   NO   NO   NO   NO   NO   NO
```

[TASK]   Task Number Assignment

		Product	**Rev**
Type	`Assignment or Comparison`	6K	5.0
Syntax	`See below`		
Units	`The TASK value is the number of the controlling task.`		
Range	`0-10`		
Default	`n/a`		
Response	`n/a`		
See Also	`%, TTASK, VAR, VARI`		

The Task Number Assignment operator (`TASK`) allows the program itself to determine which task is executing it. The current task number `TASK` may be assigned to a numeric or integer variable or evaluated in a conditional statement, such as `IF` or `WAIT`.

Syntax: `VARn=TASK` or `VARIn=TASK` where "n" is the variable number; or `TASK` can be used in an expression, such as `IF(TASK=3)`.

The `TASK` operator allows a single program to be used as a subroutine called from programs running in all tasks, yet this routine could contain sections of statements which are executed by some tasks and not others. The example below demonstrates statements used to execute different `WAIT`-for-input conditions, depending on the task that is executing the program.

Example:
```
IF (TASK=1)     ; Check if this program is operating in task 1
WAIT(1IN.3=B1)  ; If in task 1, wait for input at location 3 on I/O brick 1
NIF

IF(TASK=2)      ; Check if this program is operating in task 2
WAIT(2IN.11=B1) ; If in task 2, wait for input at location 11 on I/O brick 2
NIF
```

TASX　　　　Transfer Extended Axis Status

		Product	Rev
Type	`Transfer`	6K	5.0
Syntax	`<!><a>TASX<.i>`		
Units	`i = bit location on the specified axis (See below)`		
Range	`1-32`		
Default	`n/a`		

Response
```
TASX:    *TASX 0000_0000_0000_0000_0000_0000_0000_0000
         *     0000_1000_0000_0000_0000_0000_0000_0000
         *     (repeated for each axis)
1TASX:   *1TASX0000_0000_0000_0000_0000_0000_0000_0000
         bit 1 ↰                                     ↱ bit 32
TASX.5:  *00110000 (bit 5 of all eight axes status registers)
1TASX.5: *0 (bit 5 of status register for axis 1)
```

See Also　`[AS], [ASX], [ER], EFAIL, TAS, TASXF, TER`

The Transfer Extended Axis Status (`TASX`) command returns the current status for each axis.

FULL-TEXT STATUS REPORT AVAILABLE

The `TASX` status command reports a binary bit report. If you would like to see a more descriptive text-based report, use the `TASXF` command description.

Bit Assignment (left to right)	**Function** (1 = yes, Ø = no)
1-3	RESERVED
4	Drive Fault Input Active (indicates the current hardware state of the drive fault input, <u>even if the drive and the drive fault input are disabled</u>)
5	Encoder failure (requires `EFAIL1` enabled for the axis). This bit is cleared with the `EFAILØ` command
6	Encoder Z-Channel state (`1` = active, `Ø` = inactive)
7-32	RESERVED

Bit #4 indicates the current hardware state of the drive fault input, even in the factory default power-up state —the drive is disabled (see `DRIVE` command) and the drive fault input is disabled (see `DRFEN` command).

TASXF   Transfer Extended Axis Status, (full-text report)

Type	Transfer		**Product**	**Rev**
Syntax	`<!><a>TASXF`		6K	5.0
Units	n/a			
Range	n/a			
Default	n/a			
Response	TASXF: (see example below)			
See Also	[AS], [ASX], [ER], TAS, TASX, TER			

The `TASXF` command returns a text-based status report of all axes. This is an alternative to the binary report (`TASX`). Example `TASXF` response:

```
*TASX               AXIS #
*                   1    2    3    4    5    6    7    8
*RESERVED           NO   NO   NO   NO   NO   NO   NO   NO
*RESERVED           NO   NO   NO   NO   NO   NO   NO   NO
*RESERVED           NO   NO   NO   NO   NO   NO   NO   NO
*Drive Fault Active NO   NO   NO   NO   NO   NO   NO   NO
*
*Encoder Failure    NO   NO   NO   NO   NO   NO   NO   NO
*Z-Channel Active   NO   NO   NO   NO   NO   NO   NO   NO
```

TCMDER   Transfer Command Error

Type	Transfer or Program Debug Tool		**Product**	**Rev**
Syntax	`<!>TCMDER`		6K	5.0
Units	n/a			
Range	n/a			
Default	n/a			
Response	TCMDER: *(incorrect command)			
See Also	ERRBAD, [SS], TSS			

To facilitate program debugging, the Transfer Command Error (`TCMDER`) command allows you to transfer the command that the controller detects as an error. This is especially useful if you receive an error message when running or downloading a program, because it catches and remembers the **first** command that caused the error.

When the bad command is detected, the controller sends an error message to the screen, followed by the `ERRBAD` error prompt (`?`). To determine which command is in error, enter the `TCMDER` command and the controller will display the command, including all its command fields, if any.

Once a command error has occurred, the command and its fields are stored and system status bit #11 (reported in the `TSSF`, `TSS`, and `SS` commands) is set to `1`. The status bit remains set until the `TCMDER` command is issued.

Example:
```
DEF badprg         ; Begin definition of program called badprg
MA11               ; Select the absolute preset positioning mode
A25,40             ; Set acceleration
AD11,26            ; Set deceleration
V5,8               ; Set velocity
VAR1=0             ; Set variable #1 equal to zero
GO11               ; Initiate move on both axes
IF(VAR1<)16        ; Mistyped IF statement—should be typed as: IF(VAR1<16)
VAR1=VAR1+1        ; If variable #1 is less than16, increment the counter by 1
NIF                ; End IF statement
END                ; End programming of program called badprg
RUN badprg         ; Run the program called badprg
                   ; (this will cause an error --see comment box below)
; *******************************************************************
; * 1. When you run the badprg program, you should see this error    *
; *    message on your screen: "*INCORRECT DATA"  (this error message *
; *    indicates incorrect command syntax).                          *
; * 2. Type "TCMDER" and press enter.  This queries the controller to *
; *    display the command that caused the error.  In this case, the  *
; *    response will be "*IF(VAR1<)16".                               *
; *******************************************************************
```

TDAC

Transfer Digital-to-Analog Converter (DAC) Voltage

Type	`Transfer`	**Product**	**Rev**
Syntax	`<!><@><a>TDAC`	6K	5.0
Units	Reported value represents volts		
Range	Range of reported value is –10 to +10		
Default	n/a		
Response	TDAC: *TDAC10.000,10.000,10.000,10.000 ...		
	1TDAC: *1TDAC10.000		
See Also	[DAC], DACLIM, SFB, SGAF, SGI, SGP, SGV, SGVF, SOFFS		

This command allows you to display the voltage being commanded at the digital-to-analog converter (DAC). This is the *analog command signal* (plus any voltage offset set with the `SOFFS` command) output by the servo controller. The DAC output is a 12-bit, ± 10V analog signal. At any point, the voltage that is currently being commanded can be displayed using the `TDAC` command. If direct control over the analog voltage is required, it can be accomplished by setting the servo algorithm gains (`SGP`, `SGI`, `SGV`, `SGVF`, & `SGAF`) to zero and using the `SOFFS` command.

Example:
```
TDAC        ; Display the actual output voltage for each axis.
            ; Example response is: *TDAC4.552,5.552,5.552,5.552
```

TDIR

Transfer Program Directory

Type	`Transfer`	**Product**	**Rev**
Syntax	`<!>TDIR`	6K	5.0
Units	n/a		
Range	n/a		
Default	n/a		
Response	TDIR: *NO PROGRAMS DEFINED		
	*33000 OF 33000 BYTES (100%) PROGRAM MEMORY REMAINING		
	*500 of 500 SEGMENTS (100%) COMPILED MEMORY REMAINING		
See Also	DEF, INFNC, LIMFNC, MEMORY, PLCP, [SEG] TMEM, TSEG		

The Transfer Program Directory (`TDIR`) command returns the names of all the programs and subroutines defined with the `DEF` command, and the amount of memory each consumes. The format of the response is as follows:

```
*1 – PROG1 USES 345 BYTES
*2 – PROG2 USES 333 BYTES
*32322 OF 33000 BYTES (98%) PROGRAM MEMORY REMAINING
*500 of 500 SEGMENTS (100%) COMPILED MEMORY REMAINING
```

(In the above example, `PROG1` and `PROG2` are names of programs.)

NOTE: The amount of memory available is product-dependent.

The number in front of the program name is the number to use when defining specific inputs (`INFNC`) to correspond to a specific program (function P of `INFNC` or `LIMFNC`), or when programs are selected via BCD (function B of `INFNC` or `LIMFNC`).

If the program is intended to be a compiled profile and has been successfully compiled (`PCOMP`), then the line item for a compiled contouring or `GOBUF` program is amended with "COMPILED AS A PATH", and the line item for a compiled PLCP program is "COMPILED AS A PLC PROGRAM."

TDPTR   Transfer Data Pointer Status

Type	Data Storage	**Product**	**Rev**
Syntax	`<!>TDPTR`	6K	5.0
Units	n/a		
Range	n/a		
Default	n/a		
Response	TDPTR *TDPTR1,1,1		
See Also	DATPTR, DATSIZ, [DPTR]		

The `TDPTR` command responds with a 3-integer status report (`i,i,i`). The first integer is the number of the current active data program (the program number specified with the last `DATSIZ` or `DATPTR` command). The second integer is the location number of the data element to which the data pointer is currently pointing. The third integer is the increment set with the last `DATPTR` command.

The `DPTR` command can be used to compare the current pointer location against another value or variable, or to assign the pointer location number to a variable.

Example
```
DATSIZ4,200    ; Create data program called DATP4 with 200 data elements
DATPTR4,20,2   ; Set the data pointer to data element #20 in DATP4 and set the
               ; increment to 2 (DATP4 becomes the current active data program)
TDPTR          ; Response is *TDPTR4,20,2. Indicates that the data pointer is
               ; pointing to data element #20 in data program #4 (DATP4),
               ; and the increment setting is 2.
```

TER   Transfer Error Status

Type	Transfer	**Product**	**Rev**
Syntax	`<!><%>TER<.i>`	6K	5.0
Units	i = specific error status bit (specific to the task)		
Range	1-32		
Default	n/a		
Response	TER: *TER0000_0000_0000_0000_0000_0000_0000_0000		
	bit 1 ⬆ ⬆ bit 32		
	TER.4: *0 (bit 4 of error status register for Task 0)		
See Also	[ASX], DRFLVL, EFAIL, [ER], ERROR, ESTALL, GOWHEN, INFNC, LH, LIMFNC, LS, SMPER, STRGTT, TASX, TCMDER, TERF		

The Transfer Error Status (`TER`) command returns the status of the 32 error bits. There is one error status for all axes (per Task). The `TER` status command reports a binary bit report. If you would like to see a more descriptive text-based report, use the `TERF` command description.

NOTES

- The specific error bits must be enabled by the Error Enable (`ERROR`) command before the `TER` command will provide the correct status of the error conditions.

- <u>Multi-tasking</u>: If you are using multi-tasking, be aware that each task has its own error status register. Therefore, to check a specific task's error status, you must prefix the `TER` command with the task identifier (e.g., `2%TER` to check error status for Task 2). If no task identifier is given, the `TER` response is for the task supervisor (Task 0). 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 error status bit is shown below.

Bit #	**Function** (`1` = Yes; ∅ = 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 EFAILØ to the affected axis).
18	Cable to an expansion I/O brick is disconnected, or power to the I/O brick is lost; to clear the error, reconnect the I/O brick (or restore power to the I/O brick) and issue the ERROR.18-0 command and then the ERROR.18-1 command.
19-32	RESERVED

* Stepper axes only; ** Servo axes only

When error bit 5 (Commanded Kill or Stop) of the ERROR command is enabled (ERROR.5-1), a Stop (!S) or a Kill (!K or <ctrl>K) command will cause the controller to GOSUB or GOTO to the error program (ERRORP). Within the error program the cause of the error will need to be determined. The transfer error status (TER) command can be used to determine the cause of the error. If none of the error status bits are set, the cause of the error is a commanded kill or a commanded stop. The reason for not setting a bit on this error condition is that there is no way to clear the error condition upon leaving the error program.

TERF — Transfer Error Status (full-text report)

			Product	Rev
Type	Transfer		6K	5.0
Syntax	<!><%>TERF			
Units	n/a			
Range	n/a			
Default	n/a			
Response	TERF: (see example below)			
See Also	[ASX], DRFLVL, EFAIL, [ER], ERROR, ESTALL, GOWHEN, INFNC, LH, LIMFNC, LS, SMPER, STRGTT, TASX, TCMDER, TER			

The TERF command returns a text-based status report of all axes. This is an alternative to the binary report (TER). Example TERF response:

```
*TERF               AXIS #
*                   1   2   3   4   5   6   7   8
*Stall Detected     NO  NO  NO  NO  NO  NO  NO  NO
*Hard Limit Hit     NO  NO  NO  NO  NO  NO  NO  NO
*Soft Limit Hit     NO  NO  NO  NO  NO  NO  NO  NO
*Drive Fault Active NO  NO  NO  NO  NO  NO  NO  NO
*
*RESERVED           NO  NO  NO  NO  NO  NO  NO  NO
*Kill Input Active  NO  NO  NO  NO  NO  NO  NO  NO
*User Fault Input   NO  NO  NO  NO  NO  NO  NO  NO
*Stop Input Active  NO  NO  NO  NO  NO  NO  NO  NO
*
*Enable Input OK    NO  NO  NO  NO  NO  NO  NO  NO
*Profile Impossible NO  NO  NO  NO  NO  NO  NO  NO
*Target Zone Timeout NO NO  NO  NO  NO  NO  NO  NO
*Max Position Error NO  NO  NO  NO  NO  NO  NO  NO
*
*RESERVED           NO  NO  NO  NO  NO  NO  NO  NO
*GOWHEN cond true   NO  NO  NO  NO  NO  NO  NO  NO
*RESERVED           NO  NO  NO  NO  NO  NO  NO  NO
*Bad command        NO  NO  NO  NO  NO  NO  NO  NO

*Encoder Failure    NO  NO  NO  NO  NO  NO  NO  NO
*I/O Brick Failure  NO  NO  NO  NO  NO  NO  NO  NO
```

TEX
Transfer Program Execution Status

Type	Transfer	**Product**	**Rev**
Syntax	!<%>TEX	6K	5.0
Units	n/a		
Range	n/a		
Default	n/a		
Response	!TEX: *PROGRAM NOT EXECUTING		
See Also	DEF		

The Transfer Program Execution Status (TEX) command reports the status of any programs in progress (in the specified task). If using multi-tasking, you must prefix the TEX with the task you want to check (e.g., 2%TEX).

If the program PAUL was in progress (in task 0), and within that program a loop was in progress, the response to !TEX could look like the following: *PROGRAM=PAUL COMMAND=LN LOOP COUNT=12

TFB
Transfer Position of Selected Feedback Devices

Type	Transfer	**Product**	**Rev**
Syntax	<!><@><a>TFB	6K	5.0
Units	Response is position of the selected feedback devices		
Range	n/a	(applicable only to	
Default	n/a	servo axes)	
Response	TFB *TFB+0,+0,+0,+0,+0,+0,+0,+0		
	1TFB *1TFB+0		
See Also	[ANI], CMDDIR, ENCPOL, [FB], [PANI], [PE], PSET, SCALE, SCLD, SFB, TANI, TPANI, TPCE, TPE		

Use the TFB command to return the current values of the feedback sources selected with the SFB command. If you do not change the default SFB selection, the response will indicate the encoder position.

If scaling is **not** enabled, the position values returned will be counts (encoder or analog input). If scaling is enabled (SCALE1), the values will be scaled by the SCLD value.

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

Example:
```
SFB2,1       ; Select ANI feedback on axis 1 and encoder feedback on axis 2
TFB          ; Report ANI input #1's voltage and encoder #2's position.
             ; Sample response is *TFB4.256,2.436
```

TFS
Transfer Following Status

Type	Following; Transfer	**Product**	**Rev**
Syntax	<!><a>TFS	6K	5.0
Units	n/a		
Range	n/a		
Default	n/a		
Response	TFS *TFS0000_0000_0000_0000_0000_0000		
	* 0000_0000_0000_0000_0000_0000		
	(repeated for each axis)		
	1TFS *1TFS0000_0000_0000_0000_0000_0000		
	bit 1 ⬑ ⬏ bit 24		
See Also	FGADV, FMCLEN, FMCP, FOLEN, FOLMAS, FPPEN, [FS], FSHFC, FSHFD, MC, [NMCY], [PMAS], TFSF		

The Transfer Following Status (TFS) command returns the current Following status of all axes. The response for TFS is as follows (**Note:** response is product dependent):

FULL-TEXT STATUS REPORT AVAILABLE

The TFS status command reports a binary bit report. If you would like to see a more descriptive text-based report, use the TFSF command description.

Bit Assignment (left to right)	Function (YES = `1`; NO = ∅)	
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.
	Bits 1-4 indicate the status of Following motion. They mimic the meaning and organization of Axis Status (`TAS` & `AS`) bits 1-4, except that each bit indicates the current state of the ratio, rather than the current state of the velocity.	
* 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–`MC∅`).
	Bits 5-8 indicate the status required for Following motion (i.e., a master must be assigned, Following must be enabled, the master must be moving, and for many features, the master direction must be positive).	
	Unless the master is a commanded position of another axis, it is likely that minor vibration of the master will cause bits 7-8 to toggle on and off, even if the master is nominally "at rest". These bits are meant primarily as a quick diagnosis for the absence of master motion, or master motion in the wrong direction. Many features require positive master counting to work properly.	
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.
	Bits 9-12 indicate the shift status of the follower. Shifting is super-imposed motion, but if viewed alone, can have its own status. In other words, bits 10-12 describe only the shifting portion of motion.	
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 `FMCLEN` 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 (`FMCLEN`) at least once, causing the master cycle number (`NMCY`) to increment.
	Bits 13-16 indicate the status of master cycle counting. If a Following application is taking advantage of master cycle counting, these bits provide a quick summary of some important master cycle information.	
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.
	Bit 17-18 indicate the status of master position measurement features.	
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.

TFSF Transfer Following Status (full-text report)

Type	Following; Transfer		**Product**	**Rev**
Syntax	`<!><a>TFSF`		6K	5.0
Units	n/a			
Range	n/a			
Default	n/a			
Response	TFSF: (see example below)			
See Also	FGADV, FMCLEN, FMCP, FOLEN, FOLMAS, FPPEN, [FS], FSHFC, FSHFD, MC, [NMCY], [PMAS], TFS			

The TFSF command returns a text-based status report of all axes. This is an alternative to the binary report (TFS).

Example TFSF response:

```
*TFSF                   AXIS #
*                       1    2    3    4    5    6    7    8
*Follower in Ratio Move NO   NO   NO   NO   NO   NO   NO   NO
*Ratio is Negative      NO   NO   NO   NO   NO   NO   NO   NO
*Followr Ratio Changing NO   NO   NO   NO   NO   NO   NO   NO
*Follower At Ratio      NO   NO   NO   NO   NO   NO   NO   NO
*
*FOLMAS Active          NO   NO   NO   NO   NO   NO   NO   NO
*FOLEN Active           NO   NO   NO   NO   NO   NO   NO   NO
*Master is Moving       NO   NO   NO   NO   NO   NO   NO   NO
*Master Dir Neg         NO   NO   NO   NO   NO   NO   NO   NO
*
*OK to Shift            NO   NO   NO   NO   NO   NO   NO   NO
*Shifting now           NO   NO   NO   NO   NO   NO   NO   NO
*Shift is Continuous    NO   NO   NO   NO   NO   NO   NO   NO
*Shift Dir is Neg       NO   NO   NO   NO   NO   NO   NO   NO
*
*Master Cyc Trig Arm    NO   NO   NO   NO   NO   NO   NO   NO
*Mas Cyc Len Given      NO   NO   NO   NO   NO   NO   NO   NO
*Master Cyc Pos Neg     NO   NO   NO   NO   NO   NO   NO   NO
*Master Cyc Num > 0     NO   NO   NO   NO   NO   NO   NO   NO
*
*Pos Prediction On      YES  YES  YES  YES  YES  YES  YES  YES
*Master Filtering On    NO   NO   NO   NO   NO   NO   NO   NO
*RESERVED               NO   NO   NO   NO   NO   NO   NO   NO
*RESERVED               NO   NO   NO   NO   NO   NO   NO   NO
*
*RESERVED               NO   NO   NO   NO   NO   NO   NO   NO
*RESERVED               NO   NO   NO   NO   NO   NO   NO   NO
*OK to do FGADV move    NO   NO   NO   NO   NO   NO   NO   NO
*FGADV move underway    NO   NO   NO   NO   NO   NO   NO   NO
```

TGAIN Transfer Servo Gains

Type	Transfer		**Product**	**Rev**
Syntax	`<@><a>TGAIN`		6K	5.0
Units	n/a			
Range	n/a		(applicable to servo axes only)	
Default	n/a			
Response	TGAIN: *SGP1,2,3,4 ...			
	*SGI.1,.1,0,0 ...			
	*SGV25,25,40,40 ...			
	*SGVF100,100,100,100 ...			
	*SGAF0,0,0,0 ...			
	1TGAIN: *1SGP1			
	*1SGI.1			
	*1SGV25			
	*1SGVF100			
	*1SGAF0			
See Also	SFB, SGAF, SGENB, SGI, SGILIM, SGP, SGSET, SGV, SGVF, SOFFS, TSGSET			

This command allows you to display the current value of each of the control algorithm gains (SGP, SGI, SGV, SGAF, & SGVF). Each time an individual gain is entered, the current value is updated to be that value. When a gain set is enabled with the SGENB command, the current value of each gain is set to the values saved in that particular gain set.

Example:
```
SGP5,5,10,10     ; Set the gains for the proportional gain
SGI.1,.1,0,0     ; Set the gains for the integral gain
SGV50,60,0,0     ; Set the gains for the velocity gain
SGVF5,6,10,11    ; Set the gains for the velocity feedforward gain
SGAF0,0,0,0      ; Set the gains for the acceleration feedforward gain
TGAIN            ; Display current values for all gains.  Example response:
                 ;  *SGP5,5,10,10
                 ;  *SGI.1,.1,0,0
                 ;  *SGV50,60,0,0
                 ;  *SGVF5,6,10,11
                 ;  *SGAF0,0,0,0
```

[TIM]   Current Timer Value

Type	Assignment or Comparison	**Product**	**Rev**
Syntax	See below	6K	5.0
Units	Milliseconds		
Range	Maximum count is 999,999,999 (approx. 11 days, 13 hours)		
Default	n/a		
Response	n/a		
See Also	TIMINT, TIMST, TIMSTP, TTIM		

The Current Timer Value (`TIM`) command is used to assign the timer value to a variable, or to make a comparison against another value. The value returned is in milliseconds.

Syntax: `VARx=<n%>TIM` where `x` is a numeric variable number, or `TIM` can be used in an expression such as `IF(TIM<24ØØ)`. <u>Multi-tasking</u>: If addressing the timer of a specific task, include the `n%` prefix.

Example:
```
VAR1=TIM         ; Timer value is assigned to variable 1
IF(TIM<1000)     ; If timer value is < 1000 milliseconds, do the IF statement
VAR1=TIM + 10    ; Timer value plus 10 assigned to variable 1
NIF              ; End IF statement
```

TIMINT   Timer Value to Cause Alarm Event

Type	Timer; Alarm Event	**Product**	**Rev**
Syntax	<!>TIMINT<b>,<i>	6K	5.0
Units	i = milliseconds		
Range	b = 0 (reset and start) or 1 (stop)		
	i = 0 – 999,999,999		
Default	0,0		
Response	TIMINT: *TIMINT0,0		
See Also	INTHW, [TIM], TIMST, TIMSTP, TTIM		

The `TIMINT` command sets the timer value upon which the 6K controller will trigger an Alarm Event. The time value at which the alarm event will occur is specified by the second field in the command.

NOTES

- To use `TIMINT`, you must first issue the `INTHW.21-1` command to enable checking for the alarm event.
- When using multi-tasking, this feature only works with the timer for Task zero.

The `TIMINT` command also determines if the timer is to be stopped when the value is reached, or if the timer is to be reset and started again. If the timer is to be stopped upon reaching the alarm value, a one should be specified for the first field. If the timer is to be reset and restarted upon reaching the alarm value,

a zero should be specified for the first field. By specifying a zero in the first field, an alarm will occur repeatedly.

Example:
```
INTHW.21-1     ; Enable checking for the timer-driven alarm event
TIMINT1,10000  ; Trigger alarm once after 10000 ms, do not restart the timer
TIMST0         ; Reset and start timer
```

TIMST   Start Timer

Type	Timer		**Product**	**Rev**
Syntax	`<!>TIMST<b><r>`		6K	5.0
Units	b = Enable bit			
	r = time (milliseconds) if b = 0, task # if b = 1			
Range	b = 0 (reset and start) or 1 (start from previous TIMSTP)			
	r = absolute time 0-999,999,999 if b = 0,			
	or r = task # 0-10 if b = 1			
Default	0			
Response	TIMST: No response, acts as if TIMST1 command was issued			
See Also	SSFR, [TIM], TIMINT, TIMSTP, TTIM			

The Start Timer (`TIMST`) command is used to start the timer.

- If `TIMST0`, you can start the timer at a specific time in milliseconds (e.g., `TIMST0,500`).
- If `TIMST1`, you can resume the timer (after stopping it with the `TIMSTP` command) with the value of the time of the specified task (e.g., `TIMST1,3`).

The timer resolution is 2 ms. The delay for executing `TIMST` and `TIMSTP` in combination is 4-6 ms.

If the timer is started and allowed to roll over the maximum timer count of 999,999,999 milliseconds (11 days, 13 hours, 46 minutes, 39.999 seconds), the timer will be stopped, and the value will be frozen at the maximum value.

Multi-Tasking: Each task has its own timer.

Example:
```
TIMST0            ; Reset and start timer
GO1100            ; Initiate motion on axes 1 and 2
TIMSTP            ; Stop timer
TTIM              ; Transfer time required for move
```

TIMSTP   Stop Timer

Type	Timer		**Product**	**Rev**
Syntax	`<!><%>TIMSTP`		6K	5.0
Units	n/a			
Range	n/a			
Default	n/a			
Response	n/a			
See Also	SSFR, [TIM], TIMINT, TIMST, TTIM			

The Stop Timer (`TIMSTP`) command stops the timer. This command in conjunction with the start timer (`TIMST`) command, provides a timer that can be used to time internal or external events.

The timer resolution is 2 ms. The delay for executing `TIMST` and `TIMSTP` in combination is 4-6 ms.

Multi-Tasking: Each task has its own timer.

Example:
```
TIMST0            ; Reset and start timer
GO1100            ; Initiate motion on axes 1 and 2
TIMSTP            ; Stop timer
TTIM              ; Transfer time required for move
```

TIN   Transfer Input Status

		Product	Rev
Type	`Transfer`	6K	5.0
Syntax	`<!><@><B>TIN<.i>`		
Units	`i = input number on the specified I/O brick (B) — see page 6`		
Range	`1-32 (product dependent)`		
Default	`n/a`		
Response	`TIN:    *0000_0000_0000_0000_0 (onboard trigger inputs)`		
	`1TIN    *0000_0000_0000_0000_0000_0000_0000_0000`		
	`1TIN.4:  *1 (status of I/O point 4 on I/O brick 1)`		
See Also	`[ IN ], INFNC, INLVL, TINO, TLIM`		

The Transfer Input Status (`TIN`) command returns the current status (active or inactive) of the programmable inputs. The input is *active* when it is grounded. The active level (active high or active low) for the inputs is established with the `INLVL` command. "High" means that current is flowing and no voltage is present at the input terminal; conversely, "low" means that no current is flowing and a voltage may be present at the input terminal. If the active level is set to active low (`INLVL∅` – default), the `TIN` response indicates active with a one (`1`) and inactive with a zero (`∅`). If the active level is set to active high (`INLVL1`), the `TIN` response indicates active with a zero (`∅`) and inactive with a one (`1`).

The inputs are numbered 1 to *n* from left to right (*n* is the maximum number of I/O points on the I/O brick). The amount of onboard and external inputs varies by product and number of external I/O bricks — refer to page 6 for details.

If the status of a specific input is required, use the bit select operator (`.`). For example, `1TIN.9` reports the status of the 1st I/O point on the 2nd SIM of I/O brick 1.

TINO   Transfer Other Input Status

		Product	Rev
Type	`Transfer`	6K	5.0
Syntax	`<!>TINO<.i>`		
Units	`i = number of input (see below)`		
Range	`6`		
Default	`n/a`		
Response	`TINO:    *TINO0000_0100`		
	`TINO.6:  *1 (status of input number 6 – ENABLE input)`		
See Also	`[ INO ], TINOF`		

The Transfer Other Input Status (`TINO`) command returns the status of all of the inputs not covered by the `TLIM` or `TIN` commands. These 8 additional inputs may be used for status feedback.

```
TINO response:  *TINObbbb_bbbb
                     ^        ^
                   Bit #1   Bit #12
```

FULL-TEXT STATUS REPORT AVAILABLE

The `TINO` status command reports a binary bit report. If you would like to see a more descriptive text-based report, use the `TINOF` command description.

Bit	Function	Location
1-5	RESERVED	
6	Enable input (`1` = OK for motion)	"ENABLE" terminal
7-8	RESERVED	

TINOF   Transfer Other Input Status (full-text report)

Type	Transfer		
Syntax	`<!>TINOF`	**Product**	**Rev**
Units	n/a	6K	5.0
Range	n/a		
Default	n/a		
Response	TINOF: (see example below)		
See Also	[INO], TINO		

The `TINOF` command returns a text-based status report of all axes. This is an alternative to the binary report (`TINO`).

Example response:

```
*TINOF
*Enable input OK     YES
```

TIO   Transfer Current Expansion I/O Status

Type	Transfer		
Syntax	`<!><B>TIO`	**Product**	**Rev**
Units	B = I/O brick number	6K	5.0
Range	1-8		
Default	n/a		
Response	(see example below)		
See Also	TIN, TINO, TLIM, TOUT, TANI		

The `TIO` command displays the status of the current I/O configuration for the controller's expansion I/O bricks. If an I/O brick is not connected, it will not be included in the status report. Onboard I/O is not reported.

The I/O bricks are connected in a series to the "EXPANSION I/O" connector (see *Installation Guide* for instructions). The 1st I/O brick in the series (closest to the 6K product) is `BRICK 1`. The next is `BRICK 2`, and so on.

Each I/O brick has 4 SIM slots and can hold from 1 to 4 I/O SIM modules. A SIM slot may hold a digital input SIM, a digital output SIM, or an analog input SIM. Each SIM provides 8 inputs or outputs; therefore, each I/O brick has 32 I/O addresses, referenced as <u>absolute</u> I/O point locations:

- SIM slot 1 = I/O points 1-8
- SIM slot 2 = I/O points 9-16
- SIM slot 3 = I/O points 17-24
- SIM slot 4 = I/O points 25-32

The `TIO` response for each I/O brick is separated into four lines, one for each SIM. I/O points 1-8 represent SIM #1, 9-16 represents SIM #2, 17-24 represents SIM #3, and 25-32 represents SIM #4. When digital outputs are detected, the report also indicates whether the jumper is set to `SINKING` or `SOURCING`. When digital inputs and outputs are detected, `TIO` displays the current hardware state and programmed function (`INFNC` for inputs and `OUTFNC` for outputs). When analog inputs are detected, `TIO` reports the current voltage present on each input.

Example `TIO` responses (in this example, 2 I/O bricks are connected to the controller):

```
>TIO
*BRICK 1:  SIM Type        Status      Function
     1-8: DIGITAL INPUTS   0000_0000   AAAA_AAAA
    9-16: DIGITAL INPUTS   0000_0000   AAAA_AAAA
   17-24: DIGITAL INPUTS   0000_0000   AAAA_AAAA
   25-32: ANALOG INPUTS    0.000,0.000,0.000,0.000,0.000,0.000,0.000,0.000
*BRICK 2:  SIM Type        Status      Function
     1-8: DIGITAL OUTPUTS  0000_0000   AAAA_AAAA  -- SINKING
    9-16: DIGITAL INPUTS   0000_0000   AAAA_AAAA
   17-24: NO SIM PRESENT
   25-32: DIGITAL OUTPUTS  0000_0000   AAAA_AAAA  -- SOURCING

>1TIO
*BRICK 1:  SIM Type        Status      Function
     1-8: DIGITAL INPUTS   0000_0000   AAAA_AAAA
    9-16: DIGITAL INPUTS   0000_0000   AAAA_AAAA
   17-24: DIGITAL INPUTS   0000_0000   AAAA_AAAA
   25-32: ANALOG INPUTS    0.000,0.000,0.000,0.000,0.000,0.000,0.000,0.000

>2TIO
*BRICK 2:  SIM Type        Status      Function
     1-8: DIGITAL OUTPUTS  0000_0000   AAAA_AAAA  -- SINKING
    9-16: DIGITAL INPUTS   0000_0000   AAAA_AAAA
   17-24: NO SIM PRESENT
   25-32: DIGITAL OUTPUTS  0000_0000   AAAA_AAAA  -- SOURCING
```

TLABEL   Transfer Labels

		Product	**Rev**
Type	`Transfer`	6K	5.0
Syntax	`<!>TLABEL`		
Units	`n/a`		
Range	`n/a`		
Default	`n/a`		
Response	`TLABEL:  *NO LABELS DEFINED`		
See Also	`$`		

The Transfer Labels (`TLABEL`) command returns the names of all the labels defined with the `$` command.

The response to a `TLABEL` command if the labels `call` and `open` are defined in a program named `prog1` is as follows:

```
*CALL DEFINED IN PROGRAM PROG1
*OPEN DEFINED IN PROGRAM PROG1
```

TLIM   Transfer Limits

Type	Transfer		**Product**	**Rev**
Syntax	`<!><a>TLIM<.i>`		6K	5.0
Units	`i = limit input number`			
Range	Product dependent			
Default	n/a			
Response	TLIM: *TLIM110_110_110_110_110_110_110_110			
	TLIM.4: *0 (status of positive-direction limit input on axis 2)			
See Also	HOM, INDEB, INFNC, [LIM], LIMFNC, LIMLVL, TAS, TASF, TIN			

The Transfer Limits (TLIM) command returns the current hardware state of the dedicated limit inputs located on the "LIMITS/HOME" connector(s). This command reports the state of the limit inputs, regardless of their assigned function with the LIMFNC command. There are 3 limit inputs per axis. To determine if an end-of-travel limit has been hit, refer to the TAS or TASF command response, bits 15 through 18.

This command does not report the status of triggers or external inputs configured as limit inputs with the INFNC command. For status on such inputs, refer to the TIN command.

The TLIM value is the debounced version of the limits status (debounced with the ØINDEB value). Axis status (TAS) bits 15 and 16 reports the non-debounced version of the end-of-travel limits.

TLIM response (bits are numbered 1-24 from left to right):

TMEM   Transfer Memory Usage

Type	Transfer		**Product**	**Rev**
Syntax	`<!>TMEM`		6K	5.0
Units	n/a			
Range	n/a			
Default	n/a			
Response	TMEM: *33000 OF 33000 BYTES (100%) PROGRAM MEMORY REMAINING			
	*500 OF 500 SEGMENTS (100%) COMPILED MEMORY REMAINING			
See Also	DEF, MEMORY, PCOMP, [SEG], TDIR, TSEG			

The Transfer Memory Usage (TMEM) command returns the amount of available memory for user program storage and for storing contouring path segments. A path segment is one element of the path (e.g., PLIN3777,3777). The amount of memory available can be modified with the MEMORY command. As programs are defined (DEF) and paths are compiled (PCOMP), the amount of memory available decreases.

TNMCY Transfer Master Cycle Number

Type	Following; Transfer	**Product**	**Rev**
Syntax	`<!><a>TNMCY`	6K	5.0
Units	n/a		
Range	n/a		
Default	n/a		
Response	`TNMCY    *TNMCY0,0,0,0,0,0,0,0` `1TNMCY   *1TNMCY0`		
See Also	FMCLEN, FMCNEW, [FS], TRGFN, TFS		

The Transfer Master Cycle Number (`TNMCY`) command displays the current master cycle number for all axes, or the axis specified. The value represents the current cycle number, not the position of the master (or the follower). The master cycle number is set to zero when master cycle counting is restarted, and is incremented each time a master cycle finishes (i.e., rollover occurs). It will often correspond to the number of complete parts in a production run. This value may be used for subsequent decision making, or simply recording the cycle number corresponding to some other event.

The master must be assigned first (`FOLMAS` command) before this command will be useful.

For a complete discussion of master cycles, please refer to the Following chapter in the *Programmer's Guide*.

TNTMAC Transfer Ethernet Address

Type	Transfer; Communications Interface	**Product**	**Rev**
Syntax	`<!>TNTMAC`	6K	5.0
Units	n/a		
Range	n/a		
Default	n/a		
Response	`TMAC: *0,144,85,0,0,1`		
See Also	NTADDR, NTMASK		

The `TNTMAC` command reports the 6K product's Ethernet address.

TOUT Transfer Output Status

Type	Transfer	**Product**	**Rev**
Syntax	`<!><@><B>TOUT<.i>`	6K	5.0
Units	i = input number on the specified I/O brick (B) — see page 6		
Range	1-32 (Product dependent)		
Default	n/a		

Response	`TOUT:    *0000_0000 (onboard outputs)`
	`1TOUT    *0000_0000_0000_0000_0000_0000_0000_0000`
	`1TOUT.4: *1 (status of I/O point 4 on I/O brick 1)`		

See Also	OUT, OUTFNC, OUTLVL, TIN, TINO		

The Transfer Output Status (`TOUT`) command returns the current status (active or inactive) of the programmable outputs. The output is *active* when it is grounded. The active level (active high or active low) for the outputs is established with the `OUTLVL` command. "High" means that current is flowing and no voltage is present at the output terminal; conversely, "low" means that no current is flowing and a voltage may be present at the output terminal. If the active level is set to active low (`OUTLVLØ` – default), the `TOUT` response indicates active with a one (`1`) and inactive with a zero (`Ø`). If the active level is set to active high (`OUTLVL1`), the `TOUT` response indicates active with a zero (`Ø`) and inactive with a one (`1`).

The outputs are numbered 1 to *n* from left to right (*n* is the maximum number of I/O points on the I/O brick). The amount of onboard and external outputs varies by product and number of external I/O bricks — refer to page 6 for details.

If the status of a specific output is required, use the bit select operator (`.`). For example, `1TOUT.9` reports the status of the 1st I/O point on the 2nd SIM of I/O brick 1.

TPANI   Transfer Position of ANI Inputs

Type	`Transfer`	**Product**	**Rev**
Syntax	`<!><B>TPANI<.i>`	6K	5.0
Units	`i = location of the analog input on the I/O brick (<B>)`		
Range	`1-32 (depending on I/O brick configuration)`	(applicable only to servo axes)	
Default	`n/a`		
Response	`1TPANI:  *1TPANIxxxx_xxxx_xxxx_xxxx_xxxx_xxxx_xxxx_xxxx`		
	`1TPANI.1 *108 (position of analog input at I/O pin 1 on I/O brick 1)`		
See Also	`[ ANI ], ANIRNG, CMDDIR, [ FB ], [ PANI ], PSET, SCALE, SCLD,`		
	`SFB, TANI, TFB`		

The `TPANI` command returns the value of the ANI analog inputs as modified by scaling (`SCLD`), offset (`PSET`), and commanded direction polarity (`CMDDIR`).

The `TPANI` and `PANI` commands are designed for applications in which the ANI input is scaled and/or used as position feedback. If you are using the ANI input to monitor an analog signal, the `TANI` and `ANI` commands would be more appropriate (`TANI` and `ANI` values are measured in volts and are unaffected by scaling, polarity, or command direction).

The `TPANI` value is represented in analog-to-digital converter (ADC) units if scaling is disabled (`SCALE0`). The ADC has a 12-bit resolution, giving a range of +2047 to -2048 counts when using the full ±10V range of the analog input (205 counts/volt). If scaling is enabled (`SCALE1`), an `SCLD` scale factor of 205 (the default value when analog input feedback is selected) allows units of volts to be used.

NOTE: If you change the voltage range of the analog input (with the `ANIRNG` command), the resolution of the `PANI` response will change accordingly. The default is ±10V.

TPC   Transfer Position Commanded

Type	`Transfer`	**Product**	**Rev**
Syntax	`<!><@><a>TPC`	6K	5.0
Units	`Reported value represents distance units (scalable by SCLD)`		
Range	`Range of the reported value is ±2,147,483,648`		
Default	`n/a`		
Response	`TPC:  *TPC+0,+0,+0,+0,+0,+0,+0,+0`		
	`1TPC: *1TPC+0`		
See Also	`CMDDIR, ERES, [ PC ], [ PCC ], PSET, SCALE, SCLD, SMPER, TAS,`		
	`TFB, TPCC, TPER`		

This command allows you to display the current *commanded position* of each axis. The `TPC` value is scaled by the distance scaling factor (`SCLD`) if scaling is enabled with the `SCALE1` command.

Servo Axes: The reported value is measured in encoder or analog input (ANI) counts.
Stepper Axes: The reported value is measured in commanded counts ("motor counts").

If you issue a `PSET` command, the commanded position value will be offset by the `PSET` command value.

Servo Axes: The commanded position (`TPC`) and the actual position (`TFB`) are used in the control algorithm to calculate the position error (`TPC - TFB = TPER`) and thereby determine the corrective control signal.

Example:
```
TPC                ; Display the current commanded position for each axis:
                   ; *TPC4000,4000,4000,4000(setpoints displayed in steps)
TFB                ; Display the current actual position for each axis:
                   ; *TFB4004,4005,4004,4003 (actual positions displayed in steps)
TPER               ; Display current position error of each axis:
                   ; *TPER-4,-5,-4,-3 (error displayed in steps)
```

TPCC   Transfer Captured Commanded Position

Type	Transfer		**Product**	**Rev**
Syntax	`<!>aTPCCc`		6K	5.0
Units	a = axis #			
	c = trigger input letter (A, B or M) for axis "a"			
	(Reported value is commanded counts, scalable by SCLD)			
Range	n/a			
Default	n/a			
Response	1TPCCA: *1TPCCA+0			
See Also	CMDDIR, ENCCNT, INFNC, [PC], [PCC], [PCMS], PSET, SCALE,			
	SCLD, SFB, TFB, TPC [TRIG], TRGLOT, TTRIG			

Use the `TPCC` command to display the current captured commanded position of a specific axis, captured with the specific "trigger interrupt" input.

Trigger Input (Axis 1-4 "TRIGGERS/OUTPUTS" connector) *		Dedicated Axis	TPCC Syntax	Trigger Input (Axis 5-8 "TRIGGERS/OUTPUTS" connector) *		Dedicated Axis	TPCC Syntax
Pin 23,	Trigger 1A	1	1TPCCA	Pin 23,	Trigger 5A	5	5TPCCA
Pin 21,	Trigger 1B	1	1TPCCB	Pin 21,	Trigger 5B	5	5TPCCB
Pin 19,	Trigger 2A	2	2TPCCA	Pin 19,	Trigger 6A	6	6TPCCA
Pin 17,	Trigger 2B	2	2TPCCB	Pin 17,	Trigger 6B	6	6TPCCB
Pin 15,	Trigger 3A	3	3TPCCA	Pin 15,	Trigger 7A	7	7TPCCA
Pin 13,	Trigger 3B	3	3TPCCB	Pin 13,	Trigger 7B	7	7TPCCB
Pin 11,	Trigger 4A	4	4TPCCA	Pin 11,	Trigger 8A	8	8TPCCA
Pin 9,	Trigger 4B	4	4TPCCB	Pin 9,	Trigger 8B	8	8TPCCB

* The number of trigger inputs available varies by product (refer to your product's *Installation Guide*).

To report an axis position captured with the MASTER TRIG input, use `aPCCM`, where "a" can be any axis number.

About Position Capture: The commanded position can be captured only by a trigger input that is defined as "trigger interrupt" input with the `INFNCi-H` command (see `INFNC` for details). Each trigger input, when configured as a "trigger interrupt" input, is dedicated to capture the position of a specific axis (see table above). When a "trigger interrupt" input is activated, the commanded position of the dedicated axis is captured and the position is available through the use of the `PCC` operator and the `TPCC` display command.

Note for Stepper Axes: By default, stepper axes capture only the commanded position. However, if the axis has Encoder Capture Mode enabled with the `ENCCNT` command, only the encoder position is captured.

Position Capture Status, Longevity of Captured Position: Use the `TTRIG` and `TRIG` commands to ascertain if a trigger interrupt input has been activated. `TTRIG` displays the status as a binary report, and `TRIG` is an assignment/comparison operator for using the status information in a conditional expression (e.g., in an `IF` statement). Once the captured commanded position value is displayed with the `TPCC` command, the `TTRIG`/`TRIG` status bit for that trigger input is cleared; but the position information remains available until it is overwritten by a subsequent position capture from the same trigger input.

Position Capture Accuracy: The commanded position capture accuracy is ±1 count.

Scaling and Position Offset: If scaling is enabled (`SCALE1`), the commanded position is scaled by the distance scaling factor (`SCLD`). If scaling is not enabled (`SCALE0`), the value reported will be actual commanded counts. If you issue a `PSET` (establish absolute position reference) command, any previously captured commanded positions will be offset by the `PSET` command value.

Example:
```
1TPCCA    ; Report axis 1's captured command position, which was captured
          ; when the dedicated trigger (TRG-1A) was activated
3TPCCB    ; Report axis 3's captured command position, which was captured
          ; when the dedicated trigger (TRG-3B) was activated
2TPCCM    ; Report axis 2's captured command position, which was captured
          ; when the master trigger (TRG-M) was activated
```

TPCE   Transfer Position of Captured Encoder

Type	Transfer		**Product**	**Rev**
Syntax	`<!>aTPCEc`		6K	5.0
Units	a = axis #			
	c = trigger input letter (A, B or M) for axis "a"			
	(Reported value represents encoder counts, scalable by SCLD)			
Range	n/a			
Default	n/a			
Response	`1TPCEA:  *1TPCEA+0`			
See Also	CMDDIR, ENCCNT, ENCPOL, INFNC, [PCE], PESET, PSET, SCALE, SCLD, SFB, TPE			

Use the `TPCE` command to display the current captured encoder position, from the time of the last trigger interrupt.

Trigger Input (Axis 1-4 "TRIGGERS/OUTPUTS" connector) *		Dedicated Axis	TPCE Syntax	Trigger Input (Axis 5-8 "TRIGGERS/OUTPUTS" connector) *		Dedicated Axis	TPCE Syntax
Pin 23,	Trigger 1A	1	1TPCEA	Pin 23,	Trigger 5A	5	5TPCEA
Pin 21,	Trigger 1B	1	1TPCEB	Pin 21,	Trigger 5B	5	5TPCEB
Pin 19,	Trigger 2A	2	2TPCEA	Pin 19,	Trigger 6A	6	6TPCEA
Pin 17,	Trigger 2B	2	2TPCEB	Pin 17,	Trigger 6B	6	6TPCEB
Pin 15,	Trigger 3A	3	3TPCEA	Pin 15,	Trigger 7A	7	7TPCEA
Pin 13,	Trigger 3B	3	3TPCEB	Pin 13,	Trigger 7B	7	7TPCEB
Pin 11,	Trigger 4A	4	4TPCEA	Pin 11,	Trigger 8A	8	8TPCEA
Pin 9,	Trigger 4B	4	4TPCEB	Pin 9,	Trigger 8B	8	8TPCEB

*The number of trigger inputs available varies by product (refer to your product's *Installation Guide*).

To report an axis position captured with the MASTER TRIG input, use `aPCEM`, where "a" can be any axis number.

About Position Capture: The encoder position can be captured only by a trigger input that is defined as "trigger interrupt" input with the `INFNCi-H` command (see `INFNC` command). Each trigger input, when configured as a "trigger interrupt" input, is dedicated to capture the position of a specific axis (see table above). When a "trigger interrupt" input is activated, the encoder position of the dedicated axis is captured and the position is available through the use of the `PCE` operator and the `TPCE` display command. **Stepper Axes**: By default, stepper axes capture only the commanded position. To capture the encoder position, the axis must be in the Encoder Capture Mode (see `ENCCNT` command).

Position Capture Status, Longevity of Captured Position: Use the `TTRIG` and `TRIG` commands to ascertain if a trigger interrupt input has been activated. `TTRIG` displays the status as a binary report, and `TRIG` is an assignment/comparison operator for using the status information in a conditional expression (e.g., in an `IF` statement). Once the captured encoder position value is reported with the `TPCE` command, the `TTRIG`/`TRIG` status bit for that trigger input is cleared; but the position information remains available until it is overwritten by a subsequent position capture from the same trigger input.

Position Capture Accuracy: The encoder position capture accuracy is ±1 encoder count.

Scaling and Position Offset: If scaling is enabled (`SCALE1`), the encoder position is scaled by the distance scaling factor (`SCLD`). If scaling is not enabled (`SCALE0`), the value reported will be actual encoder counts. If you issue a `PSET` (establish absolute position reference) command, any previously captured encoder positions will be offset by the `PSET` command value.

Example:
```
1TPCEA    ; Report axis 1's captured encoder position, which was captured
          ; when the dedicated trigger (TRG-1A) was activated
3TPCEB    ; Report axis 3's captured encoder position, which was captured
          ; when the dedicated trigger (TRG-3B) was activated
2TPCEM    ; Report axis 2's captured encoder position, which was captured
          ; when the master trigger (TRG-M) was activated
```

TPCME

Transfer Captured Master Encoder Position

Type	`Transfer`		**Product**	**Rev**
Syntax	`<!>TPCME`		6K	5.0
Units	`n/a`			
Range	`n/a`			
Default	`n/a`			
Response	`TPCME    *TPCME+0`			
See Also	`INFNC, MEPOL, MESND, [ PME ], [ PCME ], [ PCMS ], PMECLR,`			
	`PMESET, TPME, TPCMS`			

Use the `TPCME` command to display the current captured master encoder position. The master encoder is connected to the connector labeled "Master Encoder."

> **Syntax**: `VARn=PCME` where `n` is the variable number; or `PCME` can be used in an expression such as `IF(PCME>2345Ø)`.

About Position Capture: The master encoder position can be captured only by the Master Trigger input (labeled "MASTER TRIG"), and only when that input is defined as a "trigger interrupt" input with the `INFNC17-H` command (see `INFNC` command). When the "trigger interrupt" input is activated (active edge), the master encoder position is captured and the position is available through the use of the `PCME` operator and the `TPCME` display command.

Position Capture Status, Longevity of Captured Position: Use the `TTRIG` and `TRIG` commands to ascertain if a trigger interrupt input has been activated. `TTRIG` displays the status as a binary report, and `TRIG` is an assignment/comparison operator for using the status information in a conditional expression (e.g., in an `IF` statement). Once the captured master encoder position value is displayed with the `TPCME` command, `TTRIG`/`TRIG` status bit #17 is cleared; but the position information remains available until it is overwritten by a subsequent position capture from the master trigger input.

Position Capture Accuracy: The master encoder position capture accuracy is ±1 encoder count.

Scaling and Position Offset: The `TPCME` value is always in master encoder counts; it is never scaled. If you issue a `PMESET` (establish absolute position reference) command, any previously captured master encoder positions will be offset by the `PMESET` command value.

TPCMS   Transfer Captured Master Cycle Position

		Product	Rev
Type	Transfer	6K	5.0
Syntax	`<!>aTPCMSc`		
Units	a = axis #		
	c = trigger input letter (A, B or M) for axis "a"		
	(Reported value represents master counts, scalable by SCLMAS)		
Range	n/a		
Default	n/a		
Response	1TPCMSA *1TPCMSA+0		
See Also	CMDDIR, ENCCNT, ENCPOL, FOLMAS, INFNC, [PCMS], PSET, SCALE,		
	SCLMAS, SFB, TPCMS, [TRIG], TRGLOT, TTRIG		

The TPCMS command transfers the captured position of the master within its current master cycle.

TPCMS (and TPMAS) is unique among position transfers, because its value rolls over to zero each time the entire master cycle length (FMCLEN) has been traveled. Thus, the captured TPCMS value is essentially a snap-shot of the position relative to the master cycle at the time of the capture.

The master must be assigned first (FOLMAS command) before this command will be useful.

For a complete discussion of master cycles, refer to the Following chapter in the *6K Series Programmer's Guide*.

Trigger Input (Axis 1-4 "TRIGGERS/OUTPUTS" connector) *		Dedicated Axis	TPCMS Syntax	Trigger Input (Axis 5-8 "TRIGGERS/OUTPUTS" connector) *		Dedicated Axis	TPCMS Syntax
Pin 23,	Trigger 1A	1	1TPCMSA	Pin 23,	Trigger 5A	5	5TPCMSA
Pin 21,	Trigger 1B	1	1TPCMSB	Pin 21,	Trigger 5B	5	5TPCMSB
Pin 19,	Trigger 2A	2	2TPCMSA	Pin 19,	Trigger 6A	6	6TPCMSA
Pin 17,	Trigger 2B	2	2TPCMSB	Pin 17,	Trigger 6B	6	6TPCMSB
Pin 15,	Trigger 3A	3	3TPCMSA	Pin 15,	Trigger 7A	7	7TPCMSA
Pin 13,	Trigger 3B	3	3TPCMSB	Pin 13,	Trigger 7B	7	7TPCMSB
Pin 11,	Trigger 4A	4	4TPCMSA	Pin 11,	Trigger 8A	8	8TPCMSA
Pin 9,	Trigger 4B	4	4TPCMSB	Pin 9,	Trigger 8B	8	8TPCMSB

* The number of trigger inputs available varies by product (refer to your product's *Installation Guide*).

To report an axis position captured with the MASTER TRIG input, use aPCMSM, where "a" can be any axis number.

About Position Capture: The master cycle position can be captured only by a trigger input that is defined as "trigger interrupt" input with the INFNCi-H command (see INFNC command). Each trigger input, when configured as a "trigger interrupt" input, is dedicated to capture the position of a specific axis (see table above). When a "trigger interrupt" input is activated, the master cycle position of the dedicated axis is captured and the position is available through the use of the PCMS operator and the TPCMS display command.

Position Capture Status, Longevity of Captured Position: Use the TTRIG and TRIG commands to ascertain if a trigger interrupt input has been activated. TTRIG displays the status as a binary report, and TRIG is an assignment/comparison operator for using the status information in a conditional expression (e.g., in an IF statement). Once the captured master cycle position value is reported with the TPCMS command, the TTRIG/TRIG status bit for that trigger input is cleared; but the position information remains available until it is overwritten by a subsequent position capture from the same trigger input.

Position Capture Accuracy: The master cycle position is interpolated; the capture accuracy is 50 μs multiplied by the velocity of the axis at the time the trigger input was activated.

Scaling and Position Offset: If scaling is enabled (SCALE1), the master cycle position is scaled by the distance scaling factor (SCLMAS). If scaling is not enabled (SCALEØ), the value assigned will be actual counts from the commanded or encoder master source as selected with the FOLMAS command. If you issue a PSET (establish absolute position reference) command, any previously captured master cycle positions will be offset by the PSET command value.

TPE — Transfer Position of Encoder

Type	Transfer	**Product**	**Rev**
Syntax	`<!><a>TPE`	6K	5.0
Units	(Reported value represents encoder counts, scalable by SCLD)		
Range	n/a		
Default	n/a		
Response	`TPE:    *TPE+0,+0,+0,+0,+0,+0,+0,+0`		
	`1TPE:   *1TPE+0`		
See Also	CMDDIR, ENCCNT, ENCPOL, ENCSND, [FB], [PE], PESET, PSET, SCALE, SCLD, SFB, TFB		

The Transfer Position of Encoder (TPE) command returns the current encoder position. If the encoder has been configured to receive step and direction input (ENCSND), the TPE command will report the position as counted from the step and direction signal.

<u>Stepper axes</u>: If the ENCCNT1 mode is enabled TPE reports the encoder position, but in ENCCNT0 mode (the factory default setting) the TPE report represents the commanded position.

> **UNITS OF MEASURE** and **SCALING**: refer to page 16 or to the SCLD command.

If you issue a PSET command, the encoder position value will be offset by the PSET command value. If you are using a stepper axis in the ENCCNT1 mode, use the PESET command instead.

TPER — Transfer Position Error

Type	Transfers	**Product**	**Rev**
Syntax	`<!><a>TPER`	6K	5.0
Units	Reported value represents distance units (scalable by SCLD)		
Range	Range of the reported value is ±2,147,483,648	(applicable only to servo axes)	
Default	n/a		
Response	`TPER:   *TPER+0,+0,+0,+0,+0,+0,+0,+0`		
	`1TPER:  *1TPER+0`		
See Also	CMDDIR, DRES, ENCPOL, ERES, [FB], [PC], [PE], [PER], SFB, SMPER, TANI, TAS, TFB, TPE, TPC		

The Transfer Position Error (TPER) command allows you to display the current position error of each axis. The error is displayed in feedback device counts and is scaled by the distance scaling factor (SCLD), if scaling is enabled with the SCALE1 command.

The position error is the difference between the commanded position and the actual position read by the feedback device (TPER = TPC - TFB). This error is calculated every sample period and can be displayed at any time using this command.

Example:
```
TPC              ; Display the current commanded position for each axis:
                 ; *TPC4000,4000,4000,4000 (setpoints displayed in steps)
TFB              ; Display the current actual position for each axis:
                 ; *TFB4004,4005,4004,4003 (actual positions displayed in steps)
TPER             ; Display current position error of each axis:
                 ; *TPER-4,-5,-4,-3 (error displayed in steps)
```

TPMAS   Transfer Current Master Cycle Position

Type	Following; Transfer	**Product**	**Rev**
Syntax	`<!><a>TPMAS`	6K	5.0
Units	Reported value represents master counts, scalable by SCLMAS.		
Range	n/a		
Default	n/a		
Response	TPMAS *TPMAS0,0,0,0		
	1TPMAS *1TPMAS0		
See Also	FMCLEN, FMCNEW, FMCP, FOLMAS, FOLMD, [FS], MEPOL, [NMCY], [PMAS], SCALE, SCLMAS, TFS		

The TPMAS command transfers the current position of the master within its current master cycle. **The master must be assigned first (FOLMAS command) before this command will be useful.**

TPMAS is unique among position transfers, because master cycle position rolls over to zero each time the entire master cycle length (FMCLEN value) has been traveled.

If scaling is enabled (SCALE1), the value returned is scaled by the master scaling factor (SCLMAS). If scaling is disabled (SCALEØ), the value returned is in master counts (encoder counts, commanded counts, or analog input counts).

For a complete discussion of master cycles, please refer to the Following chapter in the *Programmer's Guide*.

TPME   Transfer Position of Master Encoder

Type	Transfer	**Product**	**Rev**
Syntax	`<!>TPME`	6K	5.0
Units	Reported value represents master encoder counts.		
Range	n/a		
Default	n/a		
Response	TPME *TPME+0		
See Also	MEPOL, MESND, [PCME], [PE], [PME], PMECLR, PMESET, TPCME		

Use the TPME command to display the current master encoder position. The master encoder is connected to the connector labeled "Master Encoder". If you issue a PMESET command, the master encoder position value will be offset by the PMESET command value. The TPME value is always in encoder counts, it is never scaled.

TPROG   Transfer Program Contents

Type	Transfer	**Product**	**Rev**
Syntax	`<!>TPROG<t>`	6K	5.0
Units	t = text (name of program)		
Range	Text name of 6 characters or less		
Default	n/a		
Response	n/a		
See Also	DEF, TDIR, TMEM		

The Transfer Program (TPROG) command displays the contents of the program specified. If there is no such program, then the error message *INVALID DATA will be generated. To see which programs have been created, use the TDIR command.

TPSHF   Transfer Net Position Shift

		Product	**Rev**
Type	Following; Transfer	6K	5.0
Syntax	`<!><a>TPSHF`		
Units	(Reported value represents commanded counts, scalable by SCLD)		
Range	n/a		
Default	n/a		
Response	`TPSHF    *TPSHF+0,+0,+0,+0,+0,+0,+0,+0`		
See Also	FMCNEW, FMCP, FOLEN, FSHFC, FSHFD, [PSHF], SCALE, SCLD		

The `TPSHF` command transfers the net (absolute) follower axis position shift that has occurred since that last `FOLEN1` command. The position returned will be the sum of all shifts performed on that axis, or axes, including decelerations due to limits, kill, or stop. The shift value is set to zero each time a new `FOLEN1` command or a `FOLMAS` command (with a value other than zero) is issued.

If scaling in enabled (`SCALE1`), the `PSHF` value is scaled by the distance scaling factor (`SCLD`). If scaling is not enabled, the value is in commanded counts.

TPSLV   Transfer Current Commanded Position of Follower Axis

		Product	**Rev**
Type	Following; Transfer	6K	5.0
Syntax	`<!><a>TPSLV`		
Units	(Reported value represents commanded counts, scalable by SCLD)		
Range	n/a		
Default	n/a		
Response	`TPSLV    *TPSLV+0,+0,+0,+0,+0,+0,+0,+0`		
See Also	FMCNEW, FMCP, [PSLV], SCALE, SCLD		

The `TPSLV` command transfers the current commanded position of the follower axis. The master must be assigned first (`FOLMAS` command) before this command will be useful.

If scaling in enabled (`SCALE1`), the `PSLV` value is scaled by the distance scaling factor (`SCLD`). If scaling is not enabled, the value is in commanded counts.

TRACE   Program Trace Mode Enable

		Product	**Rev**
Type	Program Debug Tool	6K	5.0
Syntax	`<!>TRACE<b>`		
Units	n/a		
Range	b = 0 (disable), 1 (enable) or X (don't care)		
Default	0		
Response	`TRACE:   *TRACE0`		
See Also	[,], [#], PORT, [SS], STEP, TRACEP, TRANS, TSS		

The Program Trace Mode Enable (`TRACE`) command enables program trace mode. When in program trace mode, all commands executed are or transferred out the Ethernet, RS-232 or RS-485 port, along with the program from which the command came.

Example:
```
DEF pick         ; Begin definition of program named pick
GO1100           ; Initiate motion on axes 1 and 2
IF(VAR1=5)       ; If variable 1 = 5 then do commands between IF and NIF
  GOTOpick1      ; Goto label pick1
  ELSE           ; Else part of IF command
  GOTOpick2      ; Goto label pick2
NIF              ; End IF command
$pick1           ; Label declaration for pick1
GO0011           ; Initiate motion on axes 3 and 4
BREAK            ; Break out of current subroutine or program
$pick2           ; Label declaration for pick2
GO1001           ; Initiate motion on axes 1 and 4
END              ; End program definition
TRACE1           ; Enable trace mode.
VAR1=5           ; Set variable 1 to 5
```

```
@LH0              ; Disable all limits
EOT13,10,0        ; Set End-of-Transmission characters to a carriage return
                  ; and a line feed
RUN pick          ; Initiate program pick
```

After executing `RUN pick`, the following information will be placed in the output buffer, due to the trace mode being enabled. (Assume variable 1 = 5)

```
*PROGRAM=PICK COMMAND=GO1100
*PROGRAM=PICK COMMAND=IF(VAR1=5.0)
*PROGRAM=PICK COMMAND=GOTO PICK1
*PROGRAM=PICK COMMAND=$PICK1
*PROGRAM=PICK COMMAND=GO0011
*PROGRAM=PICK COMMAND=BREAK
```

TRACEP   Program Flow Mode Enable

Type	`Program Debug Tool`	**Product**	**Rev**
Syntax	`<!>TRACEP<b>`	6K	5.0
Units	`n/a`		
Range	`b = 0 (disable), 1 (enable) or X (don't care)`		
Default	`0`		
Response	`TRACEP:  *TRACEP0`		
See Also	`TRACE`		

The Program Flow Mode Enable (`TRACEP`) command provides a debug tool to monitor the entry and exit of programs and their associated nest-levels.

Example:
```
DEF PICK1
GOSUB PICK2
GOTO PICK3
END

DEF PICK2
GOSUB PICK4
END

DEF PICK3
END

DEF PICK 4
END

>TRACEP1
>PICK1
*INITIATE PROGRAM:  PICK1    NEST=1
*INITIATE PROGRAM:  PICK2    NEST=2
*INITIATE PROGRAM:  PICK4    NEST=3
*END:           PROGRAM NOW:    PICK2    NEST=2
*END:           PROGRAM NOW:    PICK1    NEST=1
*INITIATE PROGRAM:  PICK3    NEST=1
*END:           PROGRAM EXECUTION TERMINATED
```

TRANS   Translation Mode Enable

Type	`Program Debug Tool`	**Product**	**Rev**
Syntax	`<!>TRANS<b>`	6K	5.0
Units	`n/a`		
Range	`b = 0 (disable), 1 (enable) or X (don't care)`		
Default	`0`		
Response	`TRANS:  *TRANS0`		
See Also	`[ # ], [ SS ], STEP, TSS`		

The Translation Mode Enable (`TRANS`) command enables the program translation mode, in which all commands processed by the 6K Series product are echoed back in their binary format (hex representation of the binary equivalent), and are not executed. The first byte (first two characters) of the response represents the command's memory requirement. The remaining bytes represent the actual command.

Example:
```
TRANS1             ; Enable translation mode
A10,20,1,1         ; Translate acceleration command A10,20,1,1. Response displayed
                   ; is: 13 01 00 00 01 86 A0 00 03 0D 40 00 00 27 10 00 00 27 10.
                   ; Note that 13 hex represents a command memory requirement of 19
                   ; bytes.
GO1100             ; Translate initiate motion command GO1100. Response displayed
                   ; is: 07 07 03 01 01 00 00.   Note that 07 hex represents a
                   ; command memory requirement of 7 bytes.
GO0011             ; Translate initiate motion command GO0011. Response displayed
                   ; is: 07 07 03 00 00 01 01.   Note that 07 hex represents a
                   ; command memory requirement of 7 bytes.
```

TREV   Transfer Revision Level

Type	`Transfer`	**Product**	**Rev**
Syntax	`<!>TREV`	6K	5.0
Units	`n/a`		
Range	`n/a`		
Default	`n/a`		
Response	`TREV: *TREV92-016740-01-5.0 (response varies by product)`		
See Also	`RESET`		

The Transfer Revision Level (`TREV`) command provides the current revision of the product's firmware. It also reports any options that have been installed. Options can be ordered through your local ATC or distributor.

TRGFN   Trigger Functions

		Product	**Rev**
Type	Inputs; Following; Motion	6K	5.0
Syntax	`<!><@>aTRGFNcbb`		
Units	a = axis #		
	c = trigger input letter for axis "a"		
	1st b = bit to select Conditional GO (GOWHEN) function		
	2nd b = bit to select Start New Master Cycle (FMCNEW) function		
Range	a = 1-8 (product dependent)		
	c = A, B, or M (M is master trigger, "TRG-M")		
	b = 0 (disable function), 1 (enable function),		
	or X (leave unchanged)		
Default	a = 1, c = A; b = 0		
Response	1TRGFN *1TRGFNA00		
See Also	[AS], ERROR, ERRORP, FMCNEW, GOWHEN, INFNC, [SS], TAS, TRGLOT, TSS, [TRIG], TTRIG		

Use the `TRGFN` command to assign certain command functions to the onboard trigger inputs. *Note that the number of trigger inputs available varies by product — see page 6.*

Trigger Input (Axis 1-4 "Triggers/Outputs" connector) *		Dedicated Axis	TRGFN Syntax	Trigger Input (Axis 5-8 "Triggers/Outputs" connector) *		Dedicated Axis	TRGFN Syntax
Pin 23,	Trigger 1A	1	1TRGFNA	Pin 23,	Trigger 5A	5	5TRGFNA
Pin 21,	Trigger 1B	1	1TRGFNB	Pin 21,	Trigger 5B	5	5TRGFNB
Pin 19,	Trigger 2A	2	2TRGFNA	Pin 19,	Trigger 6A	6	6TRGFNA
Pin 17,	Trigger 2B	2	2TRGFNB	Pin 17,	Trigger 6B	6	6TRGFNB
Pin 15,	Trigger 3A	3	3TRGFNA	Pin 15,	Trigger 7A	7	7TRGFNA
Pin 13,	Trigger 3B	3	3TRGFNB	Pin 13,	Trigger 7B	7	7TRGFNB
Pin 11,	Trigger 4A	4	4TRGFNA	Pin 11,	Trigger 8A	8	8TRGFNA
Pin 9,	Trigger 4B	4	4TRGFNB	Pin 9,	Trigger 8B	8	8TRGFNB

"Master Trigger" (TRIG-M) trigger: syntax is `TRGFNM`

* The number of trigger inputs available varies by product (refer to your product's *Installation Guide*).

NOTE

The trigger input used in this command must first be defined as a *Trigger Interrupt* input with the `INFNCi-H` command.

- "**Conditional GO**" Function (`aTRGFNc1x`): Suspend execution of the next start-motion command until the specified trigger input goes active. Start-motion commands are:
    - `GO` (standard command to begin motion)
    - `GOL` (begin linear interpolated motion)
    - `FGADV` (begin geared advance – for Following motion)
    - `FSHFC` (begin continuous shift – for Following motion)
    - `FSHFD` (begin preset shift – for Following motion)

  Axis status bit #26 (reported with `TASF`, `TAS`, or `AS`) is set to one (`1`) when there is a pending "Conditional GO" condition initiated by a `TRGFN` command; this bit is cleared when the trigger is activated or when a stop command (`S`) or a kill command (`K`) is issued. If you need execution to be triggered by other factors (e.g., input state, master position, encoder position, etc.) use the `GOWHEN` command.

- "**New Master Cycle**" Function (`aTRGFNcx1`): This is equivalent to executing the `FMCNEW` command. When the specified trigger input goes active, the controller begins a new Following master cycle. For more information on master cycles, refer to the Following chapter in the *Programmer's Guide*.

These trigger functions are cleared once the function is complete. To use the trigger to perform a `GOWHEN` function again, the `TRGFN` command must be given again.

`TRGFN` **in Compiled Motion**: When used in a compiled program, a `aTRGFNc1xx` (`GOWHEN` function) command will pause the profile in progress (motion continues at constant velocity) until the trigger is

activated to execute the next move profile. When used in a compiled profile, the TRGFN command consumes one segment of compiled memory. When used in a compiled Following profile, the TRGFN command is ignored on the reverse Following profile (i.e., when the master is moving in the opposite direction of that specified in the FOLMAS command).

Trigger Interrupt Status: The status of a trigger interrupt event is reported with the TTRIG and TRIG commands

Example: (refer also to the FOLEN examples)
```
1TRGFNBx1          ; When trigger 1B goes active, axis 1 will begin a
                   ; new master cycle
2TRGFNB1x          ; When trigger 2B goes active, axis 2 will execute the move
                   ; commanded with the GO command.
GO01               ; The move on axis 2 is commanded, but will not execute until
                   ; trigger 2B goes active.
```

TRGLOT   Trigger Interrupt Lockout Time

Type	Input	**Product**	**Rev**
Syntax	`<!>TRGLOT<r>`	6K	5.0
Units	r = time in milliseconds		
Range	0-250		
Default	24		
Response	TRGLOT: *TRGLOT24		
See Also	INDEB, INFNC, RE, REG, TIN, TRGFN, [TRIG], TTRIG		

The TRGLOT command configures the amount of time in which all "trigger interrupt" inputs (all trigger inputs configured with the INFNCi-H command) are disabled between its initial active transition and its secondary active transition. This allows rapid recognition of a trigger, but prevents subsequent bouncing of the input from causing a false position capture, registration move, or TRGFN event. The lockout time affects those triggers configured as H (trigger interrupt) with the INFNC command during those interrupt actions (registration, position capture, etc.).

The TRGLOT setting overrides the existing INDEB setting for only the trigger inputs that are assigned the "Trigger Interrupt" function.

Example:
```
INFNC1-H           ; Assign trigger 1A as a "trigger interrupt" input
TRGLOT40           ; Set lockout time for all "trigger interrupt" inputs
                   ; to be 40 milliseconds
```

[TRIG]   Trigger Interrupt 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	INFNC, [PCC], [PCE], [PCME], [PCMS], TAS, TPCC, TPCE, TPCME, TPCMS, TRGFN, TTRIG		

Use the TRIG operator to assign the Trigger Interrupt status bits to a binary variable (VARB), 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, Ø, 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 Ø through 9.

Syntax: VARBn=TRIG where "n" is the binary variable number, or TRIG can be used in an expression such as IF(TRIG=b11Ø1), or IF(TRIG=h7F)

Each Trigger Interrupt status bit indicates whether a "trigger interrupt" input has been activated to capture a position, initiate a registration move, or execute a `TRGFN` function. "Trigger Interrupt" inputs are onboard trigger inputs that have been assigned the trigger interrupt function with the `INFNCi-H` command.

Each `TTRIG` bit is cleared when the captured position value is read with the `PCC`, `PCE`, `PCME`, `PCMS`, `TPCC`, `TPCE`, `TPCME`, or `TPCMS` commands, but the position information is still available from the respective register until it is overwritten by a subsequent position capture by the same trigger input.

The function of each status bit are shown in the table below (bits are numbered from left to right). A bit that is set ("1") indicated the trigger interrupt has occurred, a "0" indicates no trigger interrupt.

TTRIG bit #	Trigger Input (Axis 1-4 "Triggers/Outputs" connector) *		Dedicated Axis	TTRIG bit #	Trigger Input (Axis 5-8 "Triggers/Outputs" connector) *		Dedicated Axis
1	Pin 23,	Trigger 1A	1	9	Pin 23,	Trigger 5A	5
2	Pin 21,	Trigger 1B	1	10	Pin 21,	Trigger 5B	5
3	Pin 19,	Trigger 2A	2	11	Pin 19,	Trigger 6A	6
4	Pin 17,	Trigger 2B	2	12	Pin 17,	Trigger 6B	6
5	Pin 15,	Trigger 3A	3	13	Pin 15,	Trigger 7A	7
6	Pin 13,	Trigger 3B	3	14	Pin 13,	Trigger 7B	7
7	Pin 11,	Trigger 4A	4	16	Pin 11,	Trigger 8A	8
8	Pin 9,	Trigger 4B	4	16	Pin 9,	Trigger 8B	8
				17	Master Trigger ("TRG-M")		Master Encoder

* The number of trigger inputs available varies by product (refer to your product's *Installation Guide*).

TSCAN   Transfer Scan Time of PLCP Program

Type	Transfer; PLC Scan Program	**Product**	**Rev**
Syntax	`<!>TSCAN`	6K	5.0
Units	Response is in increments of 2 milliseconds		
Range	2 ms - unlimited		
Default	n/a		
Response	TSCAN *TSCAN4 (4 ms corresponds to 2 system updates)		
See Also	SCANP, PLCP, EXE		

The `TSCAN` command reports the duration it takes the last PLCP program to be scanned completely. A compiled PLCP program is launched into Scan mode using the `SCANP` command. During each 2 ms system update, the PLCP program is scanned an allotted 0.5 ms window. If the PLCP program requires more than 0.5 ms to be scanned, the program will be paused and then resumed at the next system update. The value reported by the `TSCAN` command is in multiples of the 2 ms system update period.

Example:
```
SCANP PLCP1     ; Start execution of compiled PLCP program PLCP1 in Scan mode
TSCAN           ; Report the duration of the scan program in multiples of the
                ; 2 millisecond system update period. An example response might
                ; be *TSCAN4 (indicates that 2 system update periods, a total of
                ; 4 milliseconds was required to scan the last PLCP program).
```

TSEG   Transfer Number of Free Segment Buffers

Type	Compiled Motion; Transfer	**Product**	**Rev**
Syntax	`<!>TSEG`	6K	5.0
Units	n/a		
Range	n/a		
Default	n/a		
Response	TSEG: *TSEG258		
See Also	MEMORY, TDIR, TMEM, [SEG], [SS], TSS, TSSF		

The Transfer Number of Free Segment Buffers (`TSEG`) command returns the number of free segment buffers in compiled memory.

System status bit (see `TSSF`, `TSS`, and `SS`) 29 to set when the compiled memory is 75% full, and bit 30 is set if the compiled memory is 100% full.

TSGSET   Transfer Servo Gain Set

Type	Transfer		**Product**	**Rev**
Syntax	`<!>TSGSETi`		6K	5.0
Units	i = gain set identification number (see SGSET command)			
Range	1-5		(application to servo axes only)	
Default	n/a			
Response	(see examples below)			
See Also	SFB, SGAF, SGENB, SGI, SGILIM, SGP, SGSET, SGV, SGVF, SOFFS, TGAIN			

This command allows you to display any of the 5 gain sets that you saved with the SGSET command. Up to 5 gain sets can be saved.

NOTE

The tuning gains in a given gain set are specific to the feedback source that was in use (selected with the last SFB command) at the time the gains were established with the respective gain commands (SGI, SGP, etc.).

Example:
```
SGP5,5,10,10      ; Set the gain for the proportional gain
SGI.1,.1,0,0      ; Set the gain for the integral gain
SGV50,60,0,0      ; Set the gain for the velocity gain
SGVF5,6,10,11     ; Set the gain for the velocity feedforward gain
SGAF0,0,0,0       ; Set the gain for the acceleration feedforward gain
SGSET3            ; Assign the SGP, SGI, SGV, SGVF, & SGAF gains to servo gain set 3
SGP75,75,40,40    ; Set the gain for the proportional gain
SGI5,5,5,7        ; Set the gain for the integral gain
SGV1,.45,2,2      ; Set the gain for the velocity gain
SGVF0,8,0,9       ; Set the gain for the velocity feedforward gain
SGAF18,20,22,24   ; Set the gain for the acceleration feedforward gain
SGSET1            ; Assign the SGP, SGI, SGV, SGVF, & SGAF gains to servo gain set 1
SGENB1,3,3,1      ; Enable gain set 1 on axis 1 & 4 and enables gain set 3 on
                  ; axis 2 & 3
TSGSET1           ; Display gain set 1.  Response should be:
                  ;   *SGP75,75,40,40
                  ;   *SGI5,5,5,7
                  ;   *SGV1,.45,2,2
                  ;   *SGVF0,8,0,9
                  ;   *SGAF18,20,22,24
TSGSET3           ; Display gain set 3.  Response should be:
                  ;   *SGP5,5,10,10
                  ;   *SGI.1,.1,0,0
                  ;   *SGV50,60,0,0
                  ;   *SGVF5,6,10,11
                  ;   *SGAF0,0,0,0
```

TSKAX   Task Axis

Type	Multi-Tasking		**Product**	**Rev**
Syntax	`i%TASKAX<a1>,<a2>`		6K	5.0
Units	i = task number			
	a1 = first axis associated with the task			
	a2 = last axis associated with the task			
	Range of axes from a1 – a2, where a1 ≤ a2			
Range	For i, 0-10			
	For a1 and a2, 0-n,			
	where n = number of axes on the product			
Default	a1 = 1			
	a2 = n			
Response	n/a			
See Also	%, TTASK, [TASK], TSWAP, [SWAP], TSKTRN			

The Task Axis command (TSKAX) allows you to specify the axes associated with a task. The default condition in multi-tasking is that each task is associated with all controller axes. This means, for example, that when an axis being used in a task hits an end-of travel limit, program execution will be killed within that task, and in all other tasks, because they all share that axis. The TSKAX command is used to assign a

set of axes to a given task to allow a multi-axis controller to be used as more than one independent program execution environment.

The TSKAX command allows you to assign axes to specific tasks, thus constraining task response and control to a smaller set of axes. A task is allowed to control only its associated axes. This axis association covers all interaction between axes commands, conditions or inputs and task program flow. For example, if a 6K controller is controlling two independent machines that do not share common axes, with control of each machine as a separate task, a limit hit by an axis in one machine can kill the task running that machine, but will not kill the task running the other machine.

The TSKAX command allows you to specify the first and last axis numbers associated with the task. Thus, the axes associated with a task will always be consecutive. As a demonstration, the TSKAX commands in the example below will associate axes 1, 2 and 3 with Task1, axes 4, 5 and 6 with Task2, and axes 7 and 8 with Task3. If axis 3 in Task1 hits a limit, program execution in Task1 will be killed, but Task2 and Task3 can continue to run because they are independent and do not share axis 3. Task1 may change motion parameters and start motion on only axes 1, 2, and 3.

Example:
```
DEF main          ; Begin definition of program called "main"
1%TSKAX1,3        ; Associate axes 1, 2 and 3 to Task1
2%TSKAX4,6        ; Associate axes 4, 5 and 6 to Task2
3%TSKAX7,8        ; Associate axes 7 and 8 to Task3
1%move1           ; Execute stored program "move1" in Task1
2%inout           ; Execute stored program "inout" in Task2
3%fill            ; Execute stored program "fill" in Task3
END
```

It is also possible to eliminate axis association for a task altogether with the TSKAX0,0 command. This would be appropriate for a task that is not involved in motion control, but may control I/O or start other tasks.

TSKTRN   Task Turns Before Swapping

Type	Multi-Tasking		**Product**	**Rev**
Syntax	i_1%TSKTRNi_2		6K	5.0
Units	i_1 = task number			
	i_2 = number of turns before task swap			
Range	i_1 = 0-10			
	i_2 = 0-10,000			
Default	i_1 = 0			
	i_2 = 1			
Response	n/a			
See Also	%, LOCK, TTASK, [TASK], TSWAP, [SWAP], TSKAX			

Use the TSKTRN command to set the relative amount of processing time a task will get. Under default multi-tasking operation, all active tasks have an equal share of processing time; that is, each task executes one "turn" and then "swaps" control to the next active task. (A "turn" is the execution of a command, or a portion of a complex command such as those for contouring and math and trig operators.)

For example, if Task2 issued a TSKTRN6 command, while the other tasks stayed at TSKTRN1, Task2 would execute 6 commands (or portions of long commands) before relinquishing control to another task.

The TSKTRN value for a task may be changed at any time, allowing a task to increase its weight for an isolated section of program commands.

TSS — Transfer System Status

Type	`Transfer`	**Product**	**Rev**
Syntax	`<!><%>TSS<.i>`	6K	5.0
Units	`i = system status bit number`		
Range	`1-32`		
Default	`n/a`		
Response	`TSS:    *TSS1000_1000_0000_0000_0000_0000_0000_0000` `TSS.1:   *1 (status of Task 0 status bit #1—system is ready)`		
See Also	PORT, TAS, TCMDER, TRGFN, TSTAT, [TRIG], TTRIG		

The Transfer System Status (TSS) command provides information on the 32 system status bits. The TSS status command reports a binary bit report. If you would like to see a more descriptive text-based report, use the TSSF command description.

Response for TSS (b can equal Ø, 1, X, or x):   `*TSSbbbb_bbbb_bbbb_bbbb_bbbb_bbbb_bbbb_bbbb`
                         ^   ^
         Bit #1   Bit #32

MULTI-TASKING

If you are using multi-tasking, be aware that each task has its own system status register. Therefore, to check a specific task's system status, you must prefix the TSS command with the task identifier (e.g., 2%TSS to check system status for Task 2). If no task identifier is given, the TSS response is for the task supervisor (Task 0).

BIT (Left to Right)	**Function** (1 = yes, Ø = no)	**BIT** (Left to Right)	**Function** (1 = yes, Ø = no)
1	System Ready (fully powered up and ready to receive commands)	17	Loading Thumbwheel Data ([TW])
2	Reserved	18	External Program Select Mode (INSELP)
3	Executing a Program	19	Dwell in Progress (T command)
4	Immediate Command (set if last command was immediate)	20	Waiting for RP240 Data—[DREAD] or [DREADF]
5	In ASCII Mode	21	RP240 Connected— current PORT setting only
6	In Echo Mode — current PORT setting only	22	Non-volatile Memory Error
7	Defining a Program	23	Servo data gathering transmission in progress (servo axes only)
8	In Trace Mode	24	Reserved
9	In Step Mode	25	RESERVED
10	In Translation Mode (must use fast status area to see)	26	RESERVED
11	Command Error Occurred (bit is cleared when TCMDER is issued)	27	RESERVED
12	Break Point Active (BP)	28	RESERVED
13	Pause Active	29	Compiled memory is 75% full
14	Wait Active (WAIT)	30	Compiled memory is 100% full
15	Monitoring On Condition (ONCOND)	31 *	Compile operation failed (PCOMP) **
16	Waiting for Data (READ)	32	RESERVED

* Bit #31: failed PCOMP compile is cleared on power up, RESET, or after successful compile. Possible causes include:
- Errors in profile design (e.g., change direction while at non-zero velocity; distance & velocity equate to < 1 count per system update; preset move profile ends in non-zero velocity)
- Profile will cause a Following error (see TFSF, TFS, or FS command descriptions)
- Out of memory (see TSS bit #30)
- Axis already in motion at the time of the PCOMP command
- Loop programming errors (e.g., no matching PLOOP or PLN; more than 4 embedded PLOOP/END loops)
- PLCP program contains invalid commands.

TSSF   Transfer System Status (full-text report)

		Product	Rev
Type	Transfer	6K	5.0
Syntax	`<!><%>TSSF`		
Units	n/a		
Range	n/a		
Default	n/a		
Response	TSSF: (see example below)		
See Also	PORT, [SS], TAS, TCMDER, TRGFN, TSS, TSTAT		

The TSSF command returns a text-based status report of all axes. This is an alternative to the binary report (TSS).

MULTI-TASKING

If you are using multi-tasking, be aware that each task has its own system status register. Therefore, to check a specific task's system status, you must prefix the TSSF command with the task identifier (e.g., 2%TSSF to check system status for Task 2). If no task identifier is given, the TSSF response is for the task supervisor (Task 0).

Example TSSF response:

```
*TSSF
*System Ready        YES      Thumbwhl Data Load  NO
*RESERVED            NO       Ext Prog Sel Mode   NO
*Program Executing   NO       Time Command        NO
*Immediate Comm Last NO       Waiting RP240 Data  NO
*
*ASCII Mode          YES      RP240 Connected     NO
*Echo Mode           YES      Memory Error        NO
*Defining a Program  NO       Servo Data Transfer NO
*Trace Mode          NO       RESERVED            NO
*
*Step Mode           NO       RESERVED            NO
*FS Translate Mode   NO       RESERVED            NO
*Command Error       NO       RESERVED            NO
*Break Point Active  NO       RESERVED            NO
*
*Pause Active        NO       Comp Mem Near Full  NO
*Wait Active         NO       Compiled Mem Full   NO
*Checking On Conds   NO       Compile Failed      NO
*Waiting for Data    NO       Reserved            NO
```

TSTAT

Transfer Statistics

			Product	**Rev**
Type	Transfer		6K	5.0
Syntax	`<!>TSTAT`			
Units	n/a			
Range	n/a			
Default	n/a			
Response	TSTAT: (See below)			
See Also	NTADDR, TAS, TDIR, TER, TFB, TIN, TIO, TLIM, TOUT, TPC, TPE, TREV, TSKAX, TSWAP, TSS, TTIM, TUS, TVEL			

The following is an example (**NOTE:** The response for each 6K Series product will vary slightly.):

```
*6K8 (8-axis controller)
*6K revision: 92-XXXXXX-01-5.0  6K    92-XXXXXX-XX-NOP2.5 DSP
*Ethernet address: xxxxxxxxxx;  IP address: 192.168.10.30
*Axis definition: Servo,Servo,Servo,Servo,Stepper,Stepper,Stepper,Stepper
*Power-up program assignment (STARTP): SETUP
*ENABLE input OK: Yes
*Drive status (DRIVE): 0000_0000
*Drive Fault input states (ASX.4 for each axis): 0000_0000
*Drive Fault input checking - enabled (DRFEN1):  0000_0000
*Drive resolution (DRES): -,-,-,-,25000,25000,25000,25000
*Encoder resolution (ERES): 4000,4000,4000,4000
*Encoder failure detection enabled (EFAIL1): 0000_0000
*Hard Limit enable: LH3,3,3,3,3,3,3,3
*Soft Limit enable: LS0,0,0,0,0,0,0,0
*Current Motion Attributes:
*  Scaling enabled (SCALE1): 0
*  Acceleration scaler (SCLA): 4000,4000,4000,4000,4000,4000,4000,4000
*  Distance scaler (SCLD): 1,1,1,1,1,1,1,1
*  Velocity scaler (SCLV): 4000,4000,4000,4000,4000,4000,4000,4000
*  Continuous/Preset (MC1/MC0) positioning mode: 0,0,0,0,0,0,0,0
*  Absolute/Incremental (MA1/MA0) positioning mode: 0,0,0,0,0,0,0,0
*  Feedback position (TFB or TPE): +0,+0,+0,+0,-,-,-,-
*  Commanded position (TPC): +0,+0,+0,+0,+0,+0,+0,+0
*  A10.0000,10.0000,10.0000,10.0000,10.0000,10.0000,10.0000,10.0000
*  AA10.0000,10.0000,10.0000,10.0000,10.0000,10.0000,10.0000,10.0000
*  AD10.0000,10.0000,10.0000,10.0000,10.0000,10.0000,10.0000,10.0000
*  ADA10.0000,10.0000,10.0000,10.0000,10.0000,10.0000,10.0000,10.0000
*  V1.0000,1.0000,1.0000,1.0000,1.0000,1.0000,1.0000,1.0000
*  D+4000,+4000,+4000,+4000,+4000,+4000,+4000,+4000
*I/O Status:
*  Onboard limit inputs:
*    Hardware state (TLIM):   000_000_000_000_000_000_000_000
*    Prog. function (LIMFNC): RST_RST_RST_RST_RST_RST_RST_RST
*  Onboard trigger inputs:
*    Hardware state (TIN):    0000_0000_0000_0000_0
*    Prog. function (INFNC):  AAAA_AAAA_AAAA_AAAA_A
*  Onboard digital outputs:
*    Hardware state (TOUT):   000_000
*    Prog. function (OUTFNC): AAA_AAA
*  Expansion I/O bricks: See TIO response
*Axis Status (see TASF for full text report of all axes):
*  Axis 1 (1TAS): 0010_0000_0000_1000_0000_0001_0000_0000
*  Axis 2 (2TAS): 0010_0000_0000_1000_0000_0001_0000_0000
*  Axis 3 (3TAS): 0010_0000_0000_1000_0000_0001_0000_0000
*  Axis 4 (4TAS): 0010_0000_0000_1000_0000_0001_0000_0000
*  Axis 5 (5TAS): 0010_0000_0000_1000_0000_0001_0000_0000
*  Axis 6 (6TAS): 0010_0000_0000_1000_0000_0001_0000_0000
*  Axis 7 (7TAS): 0010_0000_0000_1000_0000_0001_0000_0000
*  Axis 8 (8TAS): 0010_0000_0000_1000_0000_0001_0000_0000
*System Status (This is Task 0 status if using multi-tasking.):
*    Assoc. axes (TSKAX):    1,2,3,4,5,6,7,8
*    System status (TSSF):   1000_1100_0000_0000_0000_0100_0000_0000
*    Error checking (ERROR): 1000_0100_1000_0001_0000_0000_0000_0000
*    Error status (TERF):    0000_0000_0000_0000_0000_0000_0000_0000
*    Error program (ERRORP): ERRPRG
*    On conditions (ONCOND): 0000
*Multi-Tasking Status:
*  Currently active tasks (TSWAP): 1100_0000_00
*  Task 1:
*    Assoc. axes (1%TSKAX):    1,2,3,4
*    System status (1%TSSF):   1000_1100_0000_0000_0000_0100_0000_0000
*    Error checking (1%ERROR): 1000_0100_1000_0001_0000_0000_0000_0000
*    Error status (1%TERF):    0000_0000_0000_0000_0000_0000_0000_0000
*    Error program (1%ERRORP): ERRPRG
*    On conditions (1%ONCOND): 0000
*  Task 2:
*    Assoc. axes (2%TSKAX):    5,6,7,8
*    System status (2%TSSF):   1000_1100_0000_0000_0000_0100_0000_0000
*    Error checking (2%ERROR): 1000_0100_1000_0001_0000_0000_0000_0000
*    Error status (2%TERF):    0000_0000_0000_0000_0000_0000_0000_0000
*    Error program (2%ERRORP): ERRPRG
*    On conditions (2%ONCOND): 0000
*Following Conditions:
*  Master-Follower assignment (FOLMAS): +0,+0,+0,+0,+0,+0,+0,+0
*  Master scaling (SCLMAS): 4000,4000,4000,4000,4000,4000,4000,4000
*  Following status (TFSF): 0000_0000_0000_0000_0000_0000_0000_0000
```

TSTLT   Transfer Settling Time

			Product	Rev
Type	Transfer		6K	5.0
Syntax	`<!><a>TSTLT`			
Units	Reported value represents milliseconds			
Range	n/a		(applicable only to servo axes)	
Default	n/a			
Response	`TSTLT:   *TSTLT502,483,344,249,299,443,534,674` `1TSTLT:  *1TSTLT502`			
See Also	STRGTD, STRGTE, STRGTT, STRGTV			

TSTLT allows you to display the actual time it took the last move to settle into the target zone (that is, within the distance zone defined by STRGTD and less than or equal to the velocity defined by STRGTV). The reported value represents milliseconds. **This command is usable whether or not the Target Zone Settling Mode is enabled with the STRGTE command.**

*** For a more information on target zone operation, refer to the *Programmer's Guide*.

TSWAP   Transfer Current Active Tasks

			Product	Rev
Type	Transfer		6K	5.0
Syntax	`<!>TSWAP`			
Units	n/a			
Range	Binary response status of tasks (0 = inactive, 1 = inactive). 10-bit pattern represents tasks 1-10 from left to right.			
Default	n/a			
Response	`TSWAP:    *TSWAP1001_0000_00  (tasks 1 and 4 are active)` `TSWAP.3:  *0  (task 3 is inactive)`			
See Also	%, [SS], [TASK], TSKAX, TSS			

The Transfer Task Swap command (TSWAP) reports a binary bit pattern indicating the tasks that are currently active. Note that TSWAP only indicates of a task is active; to ascertain exactly what activity the task has at a given time, use the system status (SS or TSS commands).

TSWAP's binary 10-bit pattern represents tasks 1-10, from left to right. A "1" indicates that the task is active, and a "0" indicates that the task is inactive. To check the status of only one task, you may use the bit select (.) operator. For example, TSWAP.3 checks the status of Task3 only.

The "Task Supervisor", represented by task Ø, is always active and is therefore not included in the SWAP and TSWAP status.

TTASK   Transfer Task Number

			Product	Rev
Type	Transfer		6K	5.0
Syntax	`<!>TTASK`			
Units	Reported value is the number of the controlling task.			
Range	0-10			
Default	n/a			
Response	`TTASK:   *TTASK2  (Task 2 executed the TTASK command)`			
See Also	%, [TASK]			

Use the TTASK command to the display the task number of the task which executed the command. This could be used for diagnostic purposes, as a way to indicate which task is executing a given section of program.

TTIM　　　　　Transfer Timer

		Product	Rev
Type	`Transfer`	6K	5.0
Syntax	`<!><%>TTIM`		
Units	`Reported value represents milliseconds`		
Range	`Maximum count is 999,999,999 (approx. 11 days, 13 hours)`		
Default	`n/a`		
Response	`TTIM:　　*TTIM64000`		
See Also	`T, [ TIM ], TIMINT, TIMST, TIMSTP`		

The Transfer Timer (`TTIM`) command returns the current value of the timer in milliseconds. The timer is started with the `TIMST` command, and stopped with the `TIMSTP` command.

Multi-Tasking: Each task has its own timer.

TTRIG　　　　Transfer Trigger Interrupt Status

		Product	Rev
Type	`Transfer, Inputs`	6K	5.0
Syntax	`<!>TTRIG`		
Units	`n/a`		
Range	`n/a`		
Default	`n/a`		
Response	`TTRIG　　*TTRIG0000_0000_0000_0000_0`		
See Also	`INFNC, [ PCC ], [ PCE ], [ PCME ], [ PCMS ], TAS, TPCC, TPCE,` `TPCME, TPCMS, TRGFN, [ TRIG ]`		

Use the `TTRIG` command to check whether a "trigger interrupt" input has been activated to capture a position, initiate a registration move, or execute a `TRGFN` function. "Trigger Interrupt" inputs are onboard trigger inputs that have been assigned the trigger interrupt function with the `INFNCi-H` command.

Each `TTRIG` bit is cleared when the captured position value is read with the `PCC`, `PCE`, `PCME`, `PCMS`, `TPCC`, `TPCE`, `TPCME`, or `TPCMS` commands, but the position information is still available from the respective register until it is overwritten by a subsequent position capture by the same trigger input.

The functions of each bit in the binary report are shown in the table below (bits are numbered from left to right). A bit that is set ("1") indicated the trigger interrupt has occurred, a "0" indicates no trigger interrupt.

TTRIG bit #	Trigger Input (Axis 1-4 "Triggers/Outputs" connector) *		Dedicated Axis	TTRIG bit #	Trigger Input (Axis 5-8 "Triggers/Outputs" connector) *		Dedicated Axis
1	Pin 23,	Trigger 1A	1	9	Pin 23,	Trigger 5A	5
2	Pin 21,	Trigger 1B	1	10	Pin 21,	Trigger 5B	5
3	Pin 19,	Trigger 2A	2	11	Pin 19,	Trigger 6A	6
4	Pin 17,	Trigger 2B	2	12	Pin 17,	Trigger 6B	6
5	Pin 15,	Trigger 3A	3	13	Pin 15,	Trigger 7A	7
6	Pin 13,	Trigger 3B	3	14	Pin 13,	Trigger 7B	7
7	Pin 11,	Trigger 4A	4	16	Pin 11,	Trigger 8A	8
8	Pin 9,	Trigger 4B	4	16	Pin 9,	Trigger 8B	8
				17	Master Trigger ("TRG-M")		Master Encoder

* The number of trigger inputs available varies by product (refer to your product's *Installation Guide*).

Example
```
COMEXC1                      ; Continuous command execution
INFNC1-H                     ; Define trigger 1A is a trigger interrupt input
1REGA10000                   ; Registration move on axis 1 on trigger 1A event
WAIT(TRIG.1=B1)              ; Wait for trigger 1A event to occur
WRITE"TRIGGER 1A OCCURRED"   ; Display message
TTRIG                        ; Get report back (display to monitor)
                             ; response should be: *TTRIG1000_0000_0000_0000_0
```

TUS   Transfer User Status

Type	`Transfer`	**Product**	**Rev**
Syntax	`<!>TUS<.i>`	6K	5.0
Units	`i = user status bit number`		
Range	`1 – 16`		
Default	`n/a`		
Response	`TUS:     *TUS1111_0000_1111_0000` `TUS.4:   *1 (user status bit 4 is reported)`		
See Also	`INDUSE, INDUST, [ US ]`		

The Transfer User Status (`TUS`) command returns the current bit pattern for the user status word. All 16 bits of the user status word are defined with the `INDUST` command. Each bit can correspond to an axis status bit, a system status bit, or an input.

Example:
```
INDUSE1           ; Enable user status
INDUST1-5A        ; User status bit 1 defined as axis 1 status bit 5
INDUST2-3F        ; User status bit 2 defined as axis 6 status bit 3
3INDUST3-5J       ; User status bit 3 defined as input 5 on I/O brick 3
INDUST4-1K        ; User status bit 4 defined as interrupt status bit 1
2%INDUST16-2I     ; User status bit 16 defined as system status bit 2 for task 2
TUS               ; Return the state of the user status word
```

TVEL   Transfer Current Commanded Velocity

Type	`Transfer`	**Product**	**Rev**
Syntax	`<!><a>TVEL`	6K	5.0
Units	`Reported value is in units/sec (scalable by SCLV)`		
Range	`n/a`		
Default	`n/a`		
Response	`TVEL:    *TVEL23.3450,23.0000,45.7800,456.7800 ...` `1TVEL:   *1TVEL23.3450`		
See Also	`ERES, SCALE, SCLV, TVELA, V, [ VEL ]`		

The `TVEL` value represents the current commanded velocity. It is not the programmed velocity (`V`). If scaling is enabled (`SCALE1`), the `TVEL` value is scaled by the velocity scaling factor (`SCLV`).

Stepper Axes: If scaling is disabled (`SCALEØ`), the value is measured in revolutions/sec (actual velocity in commanded counts/sec divided by the drive resolution `DRES` value).

Servo Axes: If scaling is disabled (`SCALEØ`), the value is measured in encoder revs/sec or ANI volts/sec.

TVELA   Transfer Current Actual Velocity

Type	Transfer	**Product**	**Rev**
Syntax	`<!><a>TVELA`	6K	5.0
Units	Reported value is in units/sec		
Range	n/a		
Default	n/a		
Response	TVELA: *TVELA+1.55,-3.25,-5.55,+2.30		
	1TVELA: *TVELA+1.55		
See Also	ENCCNT, SCALE, SCLV, SFB, TVEL, V, [VEL], [VELA]		

The Transfer Current Actual Velocity (TVELA) command reports the current velocity as derived from the feedback device. The sign determines the direction of motion. You can use the TVELA command at all times; therefore, even if no motion is being commanded, TVELA will still report a non-zero value as it detects the servoing action.

Units of Measure:

Steppers: The velocity is always revs/sec (actual velocity in counts/sec multiplied by the ERES value if in ENCCNT1 mode, or multiplied by DRES if in ENCCNT0 mode).

Servos: If scaling is enabled (SCALE1), the velocity value will be scaled by the velocity scaling factor (SCLV). If scaling is not enabled (SCALEØ), the value returned will be in encoder revs/sec or ANI volts/sec.

Example:
```
TVELA               ; Reports the current actual velocity; since no motion is
                    ; commanded, the servoing velocities are reported.
                    ; Example response is:  *TVELA+0.0097,-0.0027,+0.0103,-0.0044
```

TVMAS   Transfer Current Master Velocity

Type	Following and Transfer	**Product**	**Rev**
Syntax	`<!><a>TVMAS`	6K	5.0
Units	n/a		
Range	n/a		
Default	n/a		
Response	TVMAS *TVMAS+0,+0,+0,+0,+0,+0,+0,+0		
	1TVMAS *1TVMAS0		
See Also	FFILT, FOLMAS, SCALE, SCLMAS, V, [VMAS]		

The TVMAS command transfers the current velocity of the master. <u>The master must be assigned first (FOLMAS command) before this command will be useful.</u>

The precision of the reported TVMAS value is dependent upon the FFILT filter value (details are provided in the "Master Position Filtering" section in the Following chapter of the *Programmer's Guide.*

If scaling is enabled (SCALE1), the value returned is scaled by the master scaling factor (SCLMAS). If scaling is disabled (SCALE0), the value returned is in counts/sec.

[TW]   **Thumbwheel Assignment**

Type	Assignment or Comparison	**Product**	**Rev**
Syntax	TWi (See below for examples)	6K	5.0
Units	i = sets used by INPLC, INSTW, OUTPLC and OUTTW		
Range	1-8		
Default	n/a		
Response	n/a		
See Also	INPLC, INSTW, OUTPLC, OUTTW, [SS], TSS		

The Thumbwheel Assignment (TW) command, executed from within another command, reads data from a parallel device and loads it into the command field the TW command is occupying. <u>Rule of Thumb for command value substitutions</u>: If the command syntax shows that the command field requires a real number (denoted by <r>) or and integer value (denoted by <i>), you can use the TW substitution (e.g., V2,(TW)).

The value of the TW command designates which input and output set to use. TW values 1-4 correspond to INSTW and OUTTW sets 1 - 4, respectively. TW values 5-8 correspond to INPLC and OUTPLC sets 1 - 4, respectively.

The TW command can be used as a variable assignment (VAR1=TW2) or in another command (e.g., A10,(TW2),10,1). However, the TW command cannot be used in an expression such as VAR4=1 + TW2 or IF(TW2<8).

For more information on interfacing thumbwheels, refer to your product's *Installation Guide*.

Example:
```
INSTW2,1-4,5        ; Set INSTW set 2 as BCD digits on onboard inputs 1-4, with
                    ; input 5 as the sign bit
OUTTW2,1-3,4,50     ; Set OUTTW set 2 as output strobes on onboard outputs 1-3,
                    ; with output 4 as the output enable bit, and strobe time
                    ; of 50 milliseconds
A(TW2)              ; Read data into axis 1 acceleration using INSTW set 2 and
                    ; OUTTW set 2 as the data configuration
```

UNTIL( ) Until Part of Repeat Statement

			Product	**Rev**
Type	Program Flow Control		6K	5.0
Syntax	`<!>UNTIL(expression)`			
Units	n/a			
Range	Up to 80 characters (including parentheses)			
Default	n/a			
Response	n/a			
See Also	JUMP, REPEAT			

The Until Part of Repeat Statement (`UNTIL()`) command, in conjunction with the `REPEAT` command, provide a means of conditional program flow. The `REPEAT` command marks the beginning of the conditional statement. The commands between the `REPEAT` and the `UNTIL` command are executed at least once. Upon reaching the `UNTIL` command, the expression contained within the `UNTIL` command is evaluated. If the expression is false, the program flow is redirected to the first command after the `REPEAT` command. If the expression is true, the first command after the `UNTIL` command is executed.

Up to 16 levels of `REPEAT ... UNTIL()` commands may be nested.

NOTE: Be careful about performing a `GOTO` between `REPEAT` and `UNTIL`. Branching to a different location within the same program will cause the next `REPEAT` statement encountered to be nested within the previous `REPEAT` statement, unless an `UNTIL` command has already been encountered. The `JUMP` command should be used in this case.

All logical operators (AND, OR, NOT), and all relational operators (=, >, >=, <, <=, <>) can be used within the `UNTIL` expression. There is no limit on the number of logical operators, or on the number of relational operators allowed within a single `UNTIL` expression.

The limiting factor for the `UNTIL` expression is the command length. <u>The total character count for the `UNTIL` command and expression cannot exceed 80 characters.</u> For example, if you add all the letters in the `UNTIL` command and the letters within the () expression, including the parentheses and excluding the spaces, this count must be less than or equal to 80.

All assignment operators (A, AD, ANI, AS, D, DAC, DPTR, ER, IN, INO, LIM, MOV, OUT, PC, PCC, PCE, PCMS, PE, PER, SS, TIM, US, V, VEL, etc.) can be used within the `UNTIL` expression.

Example:
```
REPEAT              ; Beginning of REPEAT ... UNTIL( ) loop
GO1110              ; Initiate motion on axes 1, 2, and 3
IF(IN=b1X0)         ; IF condition: if onboard input 1 = 1, input 3 = Ø
VAR1=VAR1+1         ; If condition comes true increment variable 1 by 1
ELSE                ; Else part of IF condition
TPE                 ; If condition does not come true transfer position of
                    ; all encoders
NIF                 ; End IF statement
UNTIL(VAR1=12)      ; Repeat loop until variable 1 = 12
```

[US]　　　　　User Status

			Product	**Rev**
Type	Assignment or Comparison		6K	5.0
Syntax	See below			
Units	n/a			
Range	n/a			
Default	n/a			
Response	n/a			
See Also	INDUSE, INDUST, TUS			

The User Status (US) operator is used to assign the user 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, Ø, 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 Ø through 9.

Syntax:　VARBn=US where "n" is the binary variable number,
　　　　　　or US can be used in an expression such as IF(US=b11Ø1), or IF(US=h7)

All 16 bits of the user status word are defined with the INDUST command. Each bit can correspond to an axis status bit, a system status bit, or an input.

If it is desired to assign only one bit of the user status value to a binary variable, instead of all 16, the bit select (.) operator can be used. For example, VARB1=US.12 assigns user status bit 12 to binary variable 1.

Example:
```
VARB1=US            ; User status assigned to binary variable 1
VARB2=US.12         ; User status bit 12 assigned to binary variable 2
VARB2               ; Response, if bit 12 is set to 1, will be:
                    ;   *VARB2=XXXX_XXXX_XXX1_XXXX_XXXX_XXXX_XXXX_XXXX
IF(US=b111011X11)   ; If the user status contains 1's in bit locations
                    ; 1, 2, 3, 5, 6, 8, and 9, and a 0 in bit location 4,
                    ; do the IF statement
  TREV              ; Transfer revision level
ELSE                ; Else
  IF(US=h7F00)      ; If the user status contains 1's in bit locations
                    ; 1, 2, 3, 5, 6, 7, and 8, and 0's in every other bit
                    ; location, do the IF statement
    TSTAT           ; Transfer statistics
  NIF               ; End of second if statement
NIF                 ; End of first IF statement
```

V　　　　　Velocity

			Product	**Rev**
Type	Motion		6K	5.0
Syntax	<!><@><a>V<r>,<r>,<r>,<r>,<r>,<r>,<r>,<r>			
Units	r = units/sec			
Range	Stepper Axes:0.00000-2,048,000 (max. depends on SCLV & PULSE)			
	Servo Axes: 0.00000-6,500,000 (max. depends on SCLV)			
Default	1.0000			
Response	V: *V1.0000,1.0000,1.0000,1.0000 ...			
	1V: *1V1.0000			
See Also	GO, MC, PULSE, SCALE, SCLV, TSTAT, TVEL, TVELA, [V],			
	[VEL], [VELA], VF			

The Velocity (V) command defines the speed at which the motor will run when given a GO command. The motor will accelerate at a predefined acceleration (A) rate, before reaching the velocity (V) specified. The maximum velocity attainable is 2,048,000 units/sec (stepper axes) or 6,5000,000 units/sec (servo axes).

The velocity remains set until you change it with a subsequent velocity command. Velocities outside the valid range are flagged as an error, with a message *INVALID DATA-FIELD x, where x is the field number. When an invalid velocity is entered the previous velocity value is retained.

UNITS OF MEASURE and **SCALING**: refer to page 16.

ON-THE-FLY CHANGES: While running in the continuous mode (`MC1`), you can change velocity *on the fly* (while motion is in progress) in two ways. One way is to send an immediate velocity command (`!V`) followed by an immediate go command (`!GO`). The other, and more common, way is to enable the continuous command execution mode (`COMEXC1`) and execute a buffered velocity command (`V`) followed by a buffered go command (`GO`).

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

[V]   Velocity (Programmed) Assignment

			Product	**Rev**
Type	`Assignment or Comparison`		6K	5.0
Syntax	`See below`			
Units	`n/a`			
Range	`n/a`			
Default	`n/a`			
Response	`n/a`			
See Also	`GO, SCALE, SCLV, SSV, V, [ VEL ]`			

The velocity assignment (`V`) operator is used to compare the programmed velocity value to another value or variable, or to assign the current programmed velocity to a variable.

Syntax: `VARn=aV` where "n" is the variable number, and "a" is the axis number, or `V` can be used in an expression such as `IF(1V<25)`

When assigning the velocity value to a variable, an axis specifier must always precede the assignment (`V`) operator or it will default to axis 1 (e.g., `VAR1=1V`). When making a comparison to the programmed velocity, an axis specifier must also be used (e.g., `IF(1V<2Ø)`). The (`V`) value used in any comparison, or in any assignment statement is the programmed (`V`) value. If the actual velocity information is required, refer to the `VEL` command.

> **UNITS OF MEASURE** and **SCALING**: refer to page 16.

Example:
```
IF(2V<25)       ; If the programmed velocity on axis 2 is less than 25
                ; units/sec, then do the statements between the IF and NIF
VAR1=2V*2       ; Variable 1 = programmed velocity of axis 2 times 2
V,(VAR1)        ; Set the velocity on axis 2 to the value of variable 1
NIF             ; End the IF statement
```

VAR   Numeric Variable Assignment

Type	`Variable`		**Product**	**Rev**
Syntax	`<!>VAR<i><=r>`		6K	5.0
Units	`i = variable number`			
	`r = number or expression`			
Range	`i = 1-225`			
	`r = ±999,999,999.99999999`			
Default	`n/a`			
Response	`VAR1:    *VAR1=+0.0`			
See Also	`DVAR, VARB, VARCLR, VARI, VARS, WRVAR`			

Numeric variables can be used to store any real number value, with a range from -999,999,999.99999999 to +999,999,999.99999999. The information is assigned to the variable with the equal sign (e.g., `VAR1=32.3`).

All variables (numeric [`VAR`], integer [`VARI`], binary [`VARB`], and string [`VARS`]) are automatically stored in battery-backed RAM.

Variables are also used in conjunction with mathematical (`=`, `+`, `-`, `*`, `/`, `SQRT`), trigonometric (`ATAN`, `COS`, `PI`, `SIN`, `TAN`), and bitwise operators (`&`, `|`, `^`, `~`). For example, `VAR1=(3+4-7*4/4+3-2/1.5)*3`.

Each variable expression must be less than 80 characters in length, including the `VAR1=` part of the expression.

Numeric data can also be read into a variable, through the use of the `READ`, `DAT`, or `TW` commands (e.g., `VAR1=READ1`).

All variables can be used within commands that require a real or integer value. For example, the `A` command requires real values for acceleration; therefore, the command `A(VAR1),1Ø,12,(VAR2)` is legal. Indirect variable assignments are also legal; (e.g., `VAR(VAR1)=5` or `VAR(VAR2)=VAR(VAR4)`).

<u>Rule of Thumb for command value substitutions</u>: If the command syntax shows that the command field requires a real number (denoted by `<r>`) or and integer value (denoted by `<i>`), you can use the `VAR` substitution.

Example:
```
VAR1=2*PI            ; Set Variable 1 to 2p
D(VAR2),,(VAR3)      ; Set the distance value on axis 1 equal to variable 2,
                     ; and the distance on axis 3 equal to variable 3
```

Indirect Variables: Numeric variables can be used indirectly. Only one level of indirection is possible (e.g., `VAR(VAR(VARn))` is not a legal command). The example below shows how indirect variables are used to clear 50 variables (from 1 to 50).

Example:
```
VAR51 = 1            ; Set Variable 51 to 1
REPEAT               ; Begin repeat/until loop
VAR(VAR51) = 0       ; Clear variables (e.g., if VAR51 = 8,
                     ; then VAR(VAR51)=0 is equivalent to VAR8=0)
VAR51 = VAR51 + 1    ; Increment counter
UNTIL (VAR51 = 51)   ; End repeat/until loop
```

VARB   Binary Variable Assignment

Type	Variable		
Syntax	`<!>VARB<i><=bb...bbb>` (32 bits)	**Product**	**Rev**
Units	`i = variable number`	6K	5.0
Range	`i = 1-125`		
	`b = 0, 1, X, or x`		
Default	`n/a`		
Response	`VARB1:   *VARB1=XXXX_XXXX_XXXX_XXXX_XXXX_XXXX_XXXX_XXXX`		
See Also	`DVARB, PLCP, VAR, VARI, VARCLR, VARS, VCVT, WRVARB`		

Binary variables can be used to store any 32-bit or less binary value. The 32-bit binary value must be in the form of 32 ones, zeros, or Xs. The information is assigned to the binary variable with the equal sign.

All variables (numeric [`VAR`], integer [`VARI`], binary [`VARB`], and string [`VARS`]) are automatically stored in battery-backed RAM.

Example: `VARB1=b111100001111XXXX11110000xxxx1111`
Notice that the letter `b` is required. The `b` signifies binary, 1's, Ø's, and X's only.

Example: `VARB1=h7F4356A3`
Notice that the letter `h` is required. The `h` signifies hexadecimal, Ø-9, A-F only.

Binary variables are also used in conjunction with bitwise operators (`&, |, ^`, and `~`).
Example: `VARB1=VARB2 | VARB3 & b1111000011001`

The expression must be less than 80 characters in length, including the (`VARB1=b` or `VARB1=h`) part of the expression.

All binary variables can be used to set bits for commands that require at least 4 bits of binary information. For example, the `OUT` command requires 24 bits of binary information; therefore, the command `OUT(VARB1)` is legal.

<u>Rule of Thumb for command value substitutions</u>: If the command syntax shows that the command field requires a binary value (denoted by `<b>`), you can use the `VARB` substitution.

Example：
```
VARB1=b1110 & hA   ; Binary variable 1 is set to binary 1110 bitwise
                   ; "AND"ed with hexadecimal A
VARB1=IN.7         ; State of onboard input bit 7 assigned to binary variable 1
OUT(VARB2)         ; State of all onboard outputs assigned to binary variable 2
```

VARCLR   Variable Clear

Type	Variable		
Syntax	`<!>VARCLR`	**Product**	**Rev**
Units	`n/a`	6K	5.0
Range	`n/a`		
Default	`n/a`		
Response	`n/a`		
See Also	`DVAR, DVARI, DVARB, VAR, VARB, VARI, VARS`		

`VARCLR` resets all numeric variables (`VAR`), integer variables (`VARI`), binary variables (`VARB`), and string variables (`VARS`) to their factory default values:

> Numeric (`VAR`) and Integer (`VARI`) variables are set to `0.0`
> Binary (`VARB`) variables are set to `bxxxx_xxxx_xxxx_xxxx_xxxx_xxxx_xxxx_xxxx`
> String (`VARS`) variables are set to `""`

VARI   **Integer Variable Assignment**

		Product	**Rev**
Type	`Variable`	6K	5.0
Syntax	`<!>VARI<i><=i>`		
Units	1ˢᵗ i = variable number		
	2ⁿᵈ i = integer number		
Range	1ˢᵗ i = 1-225		
	2ⁿᵈ i = -2,147,483,647 to +2,147,483,647		
Default	`n/a`		
Response	`VARI1:   *VARI1=+Ø`		
See Also	`DVARI, PLCP, VARB, VAR, VARCLR, WRVARI`		

Integer variables can be used to store integer number value, with a range from -2,147,483,647 to +2,147,483,647. The information is assigned to the variable with the equal sign (e.g., `VARI1=32.`).

All variables (numeric [`VAR`], binary [`VARB`], integer [`VARI`], and string [`VARS`] are automatically stored in battery-backed RAM.

Integer variables can be used with mathematical (=, +, −, *, /) and bitwise operators (&, |, ^, ~). For example, `VARI1=(3+4-7*4/4+3-2/2)*3`. Numeric (`VAR`) and integer (`VARI`) variables can be mixed in the mathematical expressions. The results, if fractional, are truncated. **NOTE**: `VARI` cannot be used with trigonometric operators (`ATAN`, `COS`, `PI`, `SIN`, `TAN`) and square root (`SQRT`).

Each variable expression must be less than 80 characters in length, including the `VAR1=` part of the expression.

Numeric data can also be read into a variable, through the use of the `READ`, `DAT`, or `TW` commands (e.g., `VARI1=READ1`). Setting an integer variable to a real number results in a truncation.

All integer variables can be used within commands that require a real or integer value. For example, the `A` command requires real values for acceleration; therefore, the command `A(VARI1),1Ø,12,(VARI2)` is legal. Indirect variable assignments are also legal; (e.g., `VARI(VARI1)=5` or `VARI(VARI2)=VARI(VARI4)`).

Integer variables should be used whenever possible to allow faster math operation than the numeric variables (`VAR`).

Rule of Thumb for command value substitutions: If the command syntax shows that the command field requires a real number (denoted by <r>) or an integer value (denoted by <i>, you can use the `VARI` substitution.

Example：
```
VARI1=2*3           ; Set Variable 1 to 6
D(VARI2),,(VARI3)   ; Set the distance value on axis 1 equal to
                    ; integer variable 2, and the distance on axis 3
                    ; equal to integer variable 3
```

Indirect Variables: Integer variables can be used indirectly. Only one level of indirection is possible (e.g., `VARI(VARI(VARIn))` is not a legal command). The example below shows how indirect variables are used to clear 50 variables (from 1 to 50).

Example：
```
VARI51 = 1          ; Set Integer Variable 51 to 1
REPEAT              ; Begin repeat/until loop
VARI(VARI51) = Ø    ; Clear variables (e.g., if VARI51 = 8, then
                    ; VARI(VARI51)=Ø is equivalent to VARI8=Ø)
VARI51 = VARI51 + 1 ; Increment counter
UNTIL (VARI51 = 51)
```

VARS   **String Variable Assignment**

Type	`Variable`	**Product**	**Rev**
Syntax	`<!>VARS<i><="message">`	6K	5.0
Units	`i = variable number`		
	`message = text string`		
Range	`i = 1-25`		
	`Message = up to 20 characters`		
Default	`n/a`		
Response	`VARS1:   *VARS1="Hi John"`		
See Also	`', [ \ ], EOT, [ READ ], VAR, VARB, VARCLR, VARI, VCVT, WRITE,`		
	`WRVARS`		

String variables can be assigned a character string up to 20 characters long. The characters within the string can be any character except the quote (`"`), the semicolon (`;`), and the colon (`:`). The backslash character (`\`) immediately followed by a number is okay.

All variables (numeric [VAR], integer [VARI], binary [VARB], and string [VARS]) are automatically stored in battery-backed RAM.

To place specific control characters that are not directly available on the keyboard within a character string, use the backslash character (`\`), followed by the control character's ASCII decimal equivalent. Multiple control characters can be sent.

For example, to set the string for variable #1 equal to `HI MOM<cr>`, use the command `VARS1="HI MOM\13"` where `\13` corresponds to the carriage return character.

Common characters and their ASCII equivalent value:

Character	Description	ASCII Decimal Value
`<lf>`	Line Feed	10
`<cr>`	Carriage Return	13
`"`	Quote	34
`:`	Colon	58
`;`	Semi-colon	59
`\`	Backslash	92

Example：
```
VARS1="Enter velocity >"   ; Assign a message to string variable #1
VAR2=READ1                 ; Transmit string variable 1, and wait for numeric
                           ; data entered in the format of !'<data>.
                           ; Once numeric data is received, place it in
                           ; numeric variable 2.
                           ; Example of data entry is to type "!'10.0", which
                           ; will assign numeric variable 2 the value 10.00
```

VCVT   Variable Type Conversion

			Product	Rev
Type	`Operator (Mathematical)`		6K	5.0
Syntax	See below			
Units	n/a			
Range	n/a			
Default	n/a			
Response	n/a			
See Also	`VAR, VARB, VARI`			

Using the Variable Type Conversion (`VCVT`) operator, you can convert numeric (`VAR` or `VARI`) values to binary (`VARB`) values, and vice versa. The operation is a signed operation as the binary value is interpreted as a two's complement number, with the least significant bit (LSB) on the left and the most significant bit (MSB) on the right. A *don't care* (`X`) in a binary value will be interpreted as a zero (∅).

If the mathematical statement's result is a numeric value, then `VCVT` converts binary values to numeric values. If the statement's result is a binary value, then `VCVT` converts numeric values to binary values.

You can also convert real (`VAR`) values to integer (`VARI`) values (real values are truncated in the process).

NOTE: Numeric variables (`VAR`) have insufficient range to convert a full 32-bit binary variable (`VARB`). For example, executing the `VARB1=h00000004` command and then the `VAR1=VCVT(VARB1)` command yields an `INVALID DATA` error.

Numeric-to-Binary Conversion:
```
VAR1=-5           ; Set numeric variable value = -5
VARB1=VCVT(VAR1)  ; Convert the numeric value to a binary value and
                  ; store in VARB1
VARB1             ; Display value of VARB1.  The response should be:
                  ;   *VARB1=1101_1111_1111_1111_1111_1111_1111_1111

VAR1=25           ; Set numeric variable value = 25
VARB1=VCVT(VAR1)  ; Convert the numeric value to a binary value and
                  ; store in VARB1
VARB1             ; Display value of VARB1.  The response should be:
                  ;   *VARB1=1001_1000_0000_0000_0000_0000_0000_0000
```

Binary-to-Numeric Conversion:
```
VARB1=b0010_0110_0000_0000_0000_0000_0000_0000  ; Set binary variable = +100.0
VAR1=VCVT(VARB1)  ; Convert the binary value to a numeric value
VAR1              ; *VAR1=+100.0
```

[VEL]   Velocity (Commanded) Assignment

			Product	Rev
Type	`Assignment or Comparison`		6K	5.0
Syntax	See below			
Units	n/a			
Range	n/a			
Default	n/a			
Response	n/a			
See Also	`SCALE, SCLV, SFB, TVEL, TVELA, V, [ V ], VELA`			

Use the `VEL` operator to compare the current *commanded* velocity to another value or variable, or to assign the current commanded velocity to a variable. The velocity value used in any comparison, or in any assignment statement is the current commanded velocity value, not the *programmed* velocity (`V`) or the actual velocity as measured from the feedback device (`VELA`).

Syntax: `VARn=aVEL` where "n" is the variable number, and "a" is the axis number, or `VEL` can be used in an expression such as `IF(2VEL>4)`. When assigning the current velocity value to a variable, an axis specifier must always precede the assignment (`VEL`) operator (e.g., `VAR1=1VEL`). When making a comparison to the current velocity, an axis specifier must also be used, or else it will default to axis 1 (e.g., `IF(1VEL<2∅)`).

The VEL value represents the current commanded velocity. It is not the programmed velocity (V). If scaling is enabled (SCALE1), the VEL value is scaled by the velocity scaling factor (SCLV).

Stepper Axes: If scaling is disabled (SCALEØ), the value is measured in revolutions/sec (actual velocity in commanded counts/sec divided by the drive resolution DRES value).

Servos: If scaling is disabled (SCALEØ), the value is measured in encoder revs/sec or ANI volts/sec.

Example:
```
IF(2VEL<25)     ; If the current velocity on axis 2 is less than 25 units/sec,
                ; then do the statements between the IF and NIF
  VAR1=2V*2     ; Variable 1 = programmed velocity of axis 2 times 2
NIF             ; End the IF statement
```

[VELA]   Velocity (Actual) Assignment

			Product	Rev
Type	Assignment or Comparison			
Syntax	See below		6K	5.0
Units	n/a			
Range	n/a			
Default	n/a			
Response	n/a			
See Also	ENCCNT, ERES, SCLV, TVEL, TVELA, [V], V, [VEL]			

The VELA operator is used to compare the current *actual* velocity (as derived from the feedback device) to another value or variable, or to assign the current velocity to a variable. If the programmed velocity information is required, refer to the [V] operator; if the current commanded velocity information is required, refer to the [VEL] operator.

The sign determines the direction of motion. You can use the VELA operator at all times; therefore, even if no motion is being commanded, TVELA will still report a non-zero value as it detects the servoing action.

Syntax: VARn=aVELA where "n" is the variable number, and "a" is the axis number, or VELA can be used in an expression such as IF(2VELA>4). When assigning the current velocity value to a variable, an axis specifier must always precede the VELA assignment operator (e.g., VAR1=1VELA). When making a comparison to the current velocity, an axis specifier must also be used, or else it will default to axis 1 (e.g., IF(1VELA<2Ø)).

Units of Measure:

Steppers: The velocity is always revs/sec (actual velocity in counts/sec multiplied by the ERES value if in ENCCNT1 mode, or multiplied by DRES if in ENCCNT0 mode).

Servos: If scaling is enabled (SCALE1), the velocity value will be scaled by the velocity scaling factor (SCLV). If scaling is not enabled (SCALEØ), the value returned will be in encoder revs/sec or ANI volts/sec.

Example:
```
IF(2VELA<25)    ; If the current velocity on axis 2 is less than 25 units/sec,
                ; then do the statements between IF and NIF
  VAR1=2V*2     ; Variable 1 = programmed velocity of axis 2 times 2
NIF             ; End the IF statement
```

VF   Final Velocity

Type	Compiled Motion		**Product**	**Rev**
Syntax	`<!><@>VF<r>,<r>,<r>,<r>,<r>,<r>,<r>,<r>`		6K	5.0
Units	n/a			
Range	0 (non-zero values result in error message)			
Default	0			
Response	n/a			
See Also	FOLRN, FOLRD, FOLMAS, FOLMD, GOBUF, SCLD, FOLEN, V			

The Final Velocity (`VF`) command designates that the motor will move the load the programmed distance in a preset `GOBUF` segment, completing the move at a final speed of zero. `VF` applies only to the next (subsequent) `GOBUF`, which marks an intermediate "end of move" within a profile. `VF` is used only in conjunction with the `GOBUF` command. Normal preset `GO` moves always finish with zero velocity.

The `VF` command remains in effect for the affected axis until a `GOBUF` is executed on that axis, or until you issue a `RESET` command.

> Any non-zero value that is entered for `VF` will result in an immediate error message.

[VMAS]   Current Master Velocity

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	FFILT, FMCNEW, FMCP, FOLMAS, FOLMD, SCALE, SCLMAS, TVMAS			

The Master Velocity (`VMAS`) command is used to assign the master velocity value to a variable, or to make a comparison against another value. <u>The master must be assigned first (`FOLMAS` command) before this command will be useful.</u>

Syntax: `VARn=aVMAS` where "n" is the variable number and "a" is the axis number, or `VMAS` can be used in an expression such as `IF(2VMAS>1Ø)`. The `VMAS` command must be used with an axis specifier, or it will default to axis 1 (e.g., `VAR1=1VMAS`, `IF(2VMAS>5)`, etc.).

The precision of the `VMAS` value is dependent upon the `FFILT` filter value.

If scaling is enabled (`SCALE1`), the velocity value is scaled by the master scaling factor (`SCLMAS`). If scaling is disabled (`SCALEØ`), the velocity value is in counts/sec.

Example:
```
IF(2VMAS>4.3)    ; If the master of axis 2 is traveling at more than
                 ; 4.3 user units/sec then do the IF statement
  OUT.4=b1       ; Set onboard output #4 to 1
NIF              ; End of IF statement
VAR14=3VMAS      ; Set VAR14 to axis 3's master velocity
```

WAIT( )   **Wait for a Specific Condition**

		Product	**Rev**
Type	Program Flow Control	6K	5.0
Syntax	<!>WAIT(expression)		
Units	n/a		
Range	Up to 80 characters (including parentheses)		
Default	n/a		
Response	n/a		
See Also	FMCLEN, FMCNEW, FMCP, GOWHEN, IF, NWHILE, REPEAT, [SS], T, TSS, UNTIL, WHILE		

The Wait for a Specific Condition (WAIT) command is used to wait for a specific expression to evaluate true. No commands, except for immediate commands, after the WAIT command will be processed until the expression contained within the parentheses of the WAIT command evaluates true. The COMEXC command has no effect on the WAIT command.

All logical operators (AND, OR, NOT), and all relational operators (=, >, >=, <, <=, <>) can be used within the WAIT() expression. There is no limit on the number of logical operators, or on the number of relational operators allowed within a single WAIT() expression.

The limiting factor for the WAIT() expression is the command length. The total character count for the WAIT() command and expression cannot exceed 80 characters. For example, if you add all the letters in the WAIT command and the letters within the () expression, including the parenthesis and excluding the spaces, this count must be less than or equal to 80.

All assignment operators (A, AD, AS, D, ER, IN, INO, LIM, MOV, OUT, PC, PCE, PCMS, PE, PER, SS, TIM, US, V, VEL, etc.) can be used within the WAIT() expression.

Example:
```
MC1                ; Mode continuous
COMEXC1            ; Enable continuous command mode
GO1                ; Initiate motion on axis 1
WAIT(IN=b1)        ; Wait for onboard input 1 to be active
S1                 ; Stop motion on axis 1
WAIT(MOV=b0)       ; Wait for motion complete on axis 1
COMEXC0            ; Disable continuous command execution mode
```

WHILE( )   **WHILE Statement**

		Product	**Rev**
Type	Program Flow Control; Conditional Branching	6K	5.0
Syntax	<!>WHILE(expression)		
Units	n/a		
Range	Up to 80 characters (including parentheses)		
Default	n/a		
Response	n/a		
See Also	IF, JUMP, NWHILE, REPEAT, UNTIL		

The While Statement (WHILE) command, in conjunction with the NWHILE command, provide a means of conditional program flow. The WHILE command marks the beginning of the conditional statement, the NWHILE command marks the end. If the expression contained within the parenthesis of the WHILE command evaluates true, then the commands between the WHILE and NWHILE are executed, and continue to execute as long as the expression evaluates true. If the expression evaluates false, then program execution jumps to the first command after the NWHILE. Up to 16 levels of WHILE ... NWHILE commands may be nested.

Programming order: WHILE(expression) ...commands... NWHILE

NOTE: Be careful about performing a GOTO between WHILE and NWHILE. Branching to a different location within the same program will cause the next WHILE statement encountered to be nested within the previous WHILE statement, unless a NWHILE command has already been encountered. The JUMP command should be used in this situation.

All logical operators (AND, OR, NOT), and all relational operators (=, >, >=, <, <=, <>) can be used within the WHILE() expression. There is no limit on the number of logical operators, or on the number of relational operators allowed within a single WHILE() expression.

The limiting factor for the WHILE() expression is the command length. The total character count for the WHILE() command and expression cannot exceed 80 characters. For example, if you add all the letters in the WHILE command and the letters within the () expression, including the parenthesis and excluding the spaces, this count must be less than or equal to 80.

All assignment operators (A, AD, AS, D, ER, IN, INO, LIM, MOV, OUT, PC, PCE, PCMS, PE, PER, SS, TIM, US, V, VEL, etc.) can be used within the WHILE() expression.

Example:
```
WHILE(IN=b1X0)  ; While onboard input 1 = 1, input 3 = Ø,
                ; execute commands between WHILE and NWHILE
T5              ; Wait 5 seconds
TPE             ; Transfer position of all encoders
NWHILE          ; End WHILE statement
WHILE(1ANV<2.3) ; While analog channel 1's voltage is less than 2.3 volts,
                ; execute commands between WHILE and NWHILE
TPC             ; Transfer commanded position of all axes
NWHILE          ; End WHILE statement
```

WRITE   **Write a Message**

Type	Communication Interface	**Product**	**Rev**
Syntax	<!>WRITE"<message>"	6K	5.0
Units	n/a		
Range	Up to 69 characters (may not use ", ; or :)		
Default	n/a		
Response	WRITE"message": message		
See Also	[\ ], EOT, PORT, [READ], VARS, WRVAR, WRVARB, WRVARS		

The Write a Message (WRITE) command provides an efficient way of transmitting message strings to the Ethernet port and the RS-232C or RS-485 ports. These messages can then be used by the operating program. The EOT command characters will be transmitted after the message.

Each message can be assigned a character string up to 69 characters long. The characters within the string can be any character except the quote ("), the colon (:), and the asterisk (*).

To place specific control characters that are not directly available on the keyboard within the character string, use the backslash character (\), followed by the control character's ASCII decimal equivalent. Multiple control characters can be sent. For example, to set the message equal to HI MOM<cr>, use the command WRITE"HI MOM\13" where \13 corresponds to the carriage return character. Common characters and their ASCII equivalent values are listed below:

Character	Description	ASCII Decimal Value
<lf>	Line Feed	10
<cr>	Carriage Return	13
"	Quote	34
:	Colon	58
;	Semi-colon	59
\	Backslash	92

Example:
```
WRITE"It's a wonderful life!"   ; Send the message "It's a wonderful life!"
```

WRVAR   Write a Numeric Variable

Type	Communication Interface	Product	Rev
Syntax	`<!>WRVAR<i>`	6K	5.0
Units	i = variable number		
Range	i = 1-225		
Default	n/a		
Response	WRVAR1: +0.0		
See Also	EOT, [READ], VAR, WRITE, WRVARB, WRVARI, WRVARS		

Use the `WRVAR` command to transfer a specific numeric variable (`VAR`) to the Ethernet port and the RS-232C or RS-485 ports. Only the value and the `EOT` command characters are transmitted.

Example:
```
VAR1=100          ; Set variable 1 equal to 100
WRVAR1            ; Transmit variable 1 (the value +100.0 is transmitted)
```

WRVARB   Write a Binary Variable

Type	Communication Interface	Product	Rev
Syntax	`<!>WRVARB<i>`	6K	5.0
Units	i = variable number		
Range	i = 1-125		
Default	n/a		
Response	WRVARB1: XXXX_XXXX_XXXX_XXXX_XXXX_XXXX_XXXX_XXXX		
See Also	EOT, [READ], VARB, WRITE, WRVAR, WRVARI, WRVARS		

Use the `WRVARB` command to transfer a specific binary variable (`VARB`) to the Ethernet port and the RS-232C or RS-485 ports. Only the binary value and the `EOT` command characters are transmitted.

Example:
```
VARB1=b1101       ; Set binary variable 1 to 1101
WRVARB1           ; Transmit binary variable 1
                  ; (value transmitted =1101_XXXX_XXXX_XXXX_XXXX_XXXX_XXXX_XXXX)
```

WRVARI   Write an Integer Variable

Type	Communication Interface	Product	Rev
Syntax	`<!>WRVARI<i>`	6K	5.0
Units	i = variable number		
Range	i = 1-225		
Default	n/a		
Response	WRVARI1: +0		
See Also	EOT, [READ], VARI, WRITE, WRVAR, WRVARB, WRVARS		

Use the `WRVARI` command to transfer a specific integer variable (`VARI`) to the Ethernet port and the RS-232C or RS-485 ports. Only the integer value and the `EOT` command characters are transmitted.

Example:
```
VARI1=100         ; Set integer variable 1 equal to 100
WRVARI1           ; Transmit integer variable 1 (the value +100 is transmitted)
```

WRVARS   Write a String Variable

		Product	**Rev**
Type	Communication Interface	6K	5.0
Syntax	<!>WRVARS<i>		
Units	i = variable number		
Range	i = 1-25		
Default	n/a		
Response	WRVARS1: No response until a string is placed in VARS1		
See Also	EOT, [READ], VARS, WRITE, WRVAR , WRVARB, WRVARI		

Use the WRVARS command to transfer a specific string variable (VARS) to the Ethernet port and the RS-232C or RS-485 ports. Only the string and the EOT command characters are transmitted.

Example:
```
VARS1="John L"  ; Set string variable 1 = "John L"
WRVARS1         ; Transmit string variable 1 (string "John L" is transmitted)
```

XONOFF   Enable/Disable XON / XOFF

		Product	**Rev**
Type	Communication Interface	6K	5.0
Syntax	<!>XONOFF<b>		
Units	n/a		
Range	0 (disable), 1 (enable)		
Default	1 for COM1, 0 for COM2 (PORT command setting determines which COM port's XONOFF setting is checked)		
Response	XONOFF *XONOFF1		
See Also	], [, BOT, DRPCHK, E, EOT, ERRBAD, ERROK, LOCK, PORT		

Use the XONOFF command to enable or disable XON/XOFF (ASCII handshaking).

XONOFF1 enables XON/XOFF, which allows the 6K product to recognize ASCII handshaking control characters. When XON/XOFF is enabled, ASCII 17 or ^Q is a signal to start sending characters; ASCII 19 or ^S is a signal to stop sending characters. XONOFFØ disables XON/XOFF.

The PORT command determines which COM port is affected by the XONOFF command. Each port will track its XON/XOFF values

RS-485 Multi-drop: If you are using RS-485 multi-drop, disable XON/XOFF by executing the PORT2 command followed by the XONOFFØ command.

NOTE: COM1 is the "RS-232" connector or "ETHERNET" connector; COM2 is the "RS-232/485" connector.

Appendix A: 6K Command List

(Firmware Revision 5.0)

Command	Description	Command	Description
%	Task Identifier	COMEXL	Continue Execution on End-of-Travel Limit
[!]	Immediate Command Identifier	COMEXR	Continue Motion on Pause/Continue Input
[@]	Global Command Identifier	COMEXS	Continue Execution on Stop
;	Begin Comment	[COS( )]	Cosine [operator]
$	Label Declaration	D	Distance
[#]	Step Through a Program	[D]	Distance [operator]
'	Enter Interactive Data	[DAC]	Value of DAC Output Voltage [operator]
[.]	Bit Select	DACLIM	DAC Output Voltage Limit
["]	Begin and End String	[DAT]	Data Assignment [operator]
[\ ]	ASCII Character Designator	DATA	Data Statement
[=]	Assignment or Equivalence	[DATP]	Data Program
[>]	Greater Than	DATPTR	Set Data Pointer
[>=]	Greater Than or Equal	DATRST	Reset Data Pointer
[<]	Less Than	DATSIZ	Data Program Size
[<=]	Less Than or Equal	DATTCH	Data Teach
[<>]	Not Equal	DCLEAR	Clear RP240 Display
[( )]	Operation Priority Level	DEF	Begin Definition of Program
[+]	Addition	DEL	Delete Program
[–]	Subtraction	DJOG	Enable RP240 Jog Mode
[*]	Multiplication	[DKEY]	Value of RP240 Key
[/]	Division	DLED	Turn RP240 LEDs ON/OFF
[&]	Boolean And	DPASS	Change RP240 Password
[	]	Boolean Inclusive Or	DPCUR	Position RP240 Display Cursor
[^]	Boolean Exclusive Or	[DPTR]	Data Pointer Location [operator]
[~( )]	Boolean Not	[DREAD]	Read RP240 Numeric Data [operator]
[<<]	Shift from Right to Left (bit 32 to bit 1)	[DREADF]	Read RP240 Function Key [operator]
[>>]	Shift from Left to Right (bit 1 to bit 32)	DREADI	RP240 Data Read, Immediate Mode
[	Send Response to Both COM Ports	DRES	Drive Resolution
]	Send Response to Alternate COM Port	DRFEN	Drive Fault Input Enable
A	Acceleration	DRFLVL	Drive Fault Input, Active Level
[A]	Acceleration [operator]	DRIVE	Drive Enable/Disable
AA	Acceleration, S-curve	DRPCHK	RP240 COM Port Check
AD	Deceleration	DSTP	Enable/Disable RP240 Stop Key
[AD]	Deceleration [operator]	DVAR	Display Numeric Variable on RP240
ADA	Deceleration, S-curve	DVARB	Display Binary Variable on RP240
ADDR	Auto-Address Multiple Serial Units	DVARI	Display Integer Variable on RP240
[AND]	And [operator]	DWRITE	Write Text to RP240
[ANI]	Analog Input Voltage [operator]	E	Enable Serial Communication
ANIEN	Analog Input Enable	ECHO	Enable Communication Echo
ANIFB	Analog Inputs as Axis Feedback	EFAIL	Encoder Failure Detect
ANIMAS	Assign Analog Input as Master	ELSE	Else Condition of IF Statement
ANIRNG	Analog Input Voltage Range	ENCCNT	Encoder Count Reference Enable
[AS]	Axis Status [operator]	ENCPOL	Encoder Polarity
[ASX]	Axis Status, Extended [operator]	ENCSND	Encoder Step & Direction Mode
[ATAN( )]	Arc Tangent [operator]	END	End Definition of Program
AXSDEF	Axis Definition	EOL	End-of-Line Termination Characters
BAUD	Baud Rate	EOT	End-of-Transmission Characters
BOT	Beginning of Transmission Characters	[ER]	Error Status [operator]
BP	Set a Program Break Point	ERASE	Erase All Programs
BREAK	Terminate Program Execution	ERES	Encoder Resolution
C	Continue Command Execution	ERRBAD	Error Prompt Characters
CMDDIR	Commanded Direction Voltage	ERRDEF	Program Definition Prompt Characters
COMEXC	Continuous Command Processing Mode	ERRLVL	Error Detection Level

Command	Description	Command	Description
ERROK	Good Prompt Characters	INTSW	Force an Alarm Event
ERROR	Enable Error Checking	JOG	Enable Jog Mode
ERRORP	Assign an Error Program	JOGA	Jog Acceleration
ESDB	Stall Backlash Deadband	JOGAA	Jog Acceleration, S-curve
ESK	Kill on Stall	JOGAD	Jog Deceleration
ESTALL	Enable Stall Detection	JOGADA	Jog Deceleration, S-curve
EXE	Execute Program from a Compiled Program	JOGVH	Jog Velocity, High
[FB]	Value of Feedback Device [operator]	JOGVL	Jog Velocity, Low
FFILT	Following Filter	JOY	Enable Joystick Mode
FGADV	Following Geared Advance	JOYA	Joystick Acceleration
FMAXA	Follower Axis Maximum Acceleration	JOYAA	Joystick Acceleration, S-curve
FMAXV	Follower Axis Maximum Velocity	JOYAD	Joystick Deceleration
FMCLEN	Master Cycle Length	JOYADA	Joystick Deceleration, S-curve
FMCNEW	Restart Master Cycle Counting	JOYAXH	Joystick Analog Channel, High
FMCP	Initial Master Cycle Position	JOYAXL	Joystick Analog Channel, Low
FOLEN	Enable Following Mode	JOYCDB	Joystick Center Deadband
FOLK	Following Kill, Limitations	JOYCTR	Joystick Center
FOLMAS	Assignment of Master to Follower	JOYEDB	Joystick End Deadband
FOLMD	Master Distance	JOYVH	Joystick Velocity, High
FOLRD	Denominator of Follower-to-Master Ratio	JOYVL	Joystick Velocity, Low
FOLRN	Numerator of Follower-to-Master Ratio	JOYZ	Joystick Zero (Center)
FOLRNF	Numerator of Final Follower-to-Master Ratio	JUMP	Jump to Program or Label (No Return)
FPPEN	Enable Master Position Prediction	K	Kill Motion
[FS]	Following Status [operator]	<ctrl>K	Kill Motion
FSHFC	Continuous Shift	KDRIVE	Disable Drive on Kill
FSHFD	Preset Shift	L	Loop
FVMACC	Virtual Master Count Acceleration	LH	Enable Hardware End-of-Travel Limits
FVMFRQ	Virtual Master Count Frequency	LHAD	Hardware EOT Limits Deceleration
GO	Initiate Motion	LHADA	Hardware EOT Limits Decel, S-curve
GOBUF	Store a Compiled Motion Segment	[LIM]	Hardware EOT & Home Limit Inputs, Status
GOL	Initiate Linear Interpolated Motion	LIMEN	Limit Input Enable
GOSUB	Call a Subroutine	LIMFNC	Limit Input Function Assignment
GOTO	Goto a Program or Label	LIMLVL	Hardware EOT & Home Inputs, Active Level
GOWHEN	Conditional Go	LN	End of Loop
HALT	Terminate Program Execution	LOCK	Lock Resource to a Task
HELP	Technical Support Phone Numbers	LS	Enable Software End-of-Travel Limits
HOM	Initiate Homing Operation	LSAD	Software EOT Limits, Deceleration
HOMA	Homing Acceleration	LSADA	Software EOT Limits Decel, S-curve
HOMAA	Homing Acceleration, S-curve	LSNEG	Negative-Direction Software EOT Limit
HOMAD	Homing Deceleration	LSPOS	Positive-Direction Software EOT Limit
HOMADA	Homing Deceleration, S-curve	LX	Terminate Loop
HOMBAC	Backup to Home	MA	Enable Absolute/Incremental Positioning
HOMDF	Homing Final Direction	MC	Enable Continuous/Preset Positioning
HOMEDG	Home Reference Edge	MEMORY	Partition Product Memory
HOMV	Homing Velocity	MEPOL	Master Encoder Polarity
HOMVF	Homing Velocity, Final Approach	MESND	Master Encoder Step & Direction Mode
HOMZ	Home to Encoder Z Channel	[MOV]	Axis Moving Status [operator]
IF()	IF Statement	NIF	End IF Statement
[IN]	Input Status [operator]	[NMCY]	Master Cycle Number Status [operator]
INDEB	Input Debounce Time	[NOT]	Not [operator]
INDUSE	Enable User Status	NTADDR	Set IP Address
INDUST	User Status Definition	NTMASK	Ethernet Network Mask
INEN	Enable Specific Inputs	NWHILE	End of WHILE Statement
INFNC	Input Function Assignment	ONCOND	Enable Program Interrupt ("On") Conditions
INLVL	Input Active Level	ONIN	On an Input Condition GOSUB
[INO]	Other Inputs (Enable Input) Status [operator]	ONP	On Condition Program Assignment
INPLC	Establish PLC Data Inputs	ONUS	On a User Status Condition GOSUB
INSELP	Enable Program Selection via Inputs	ONVARA	On Numeric Variable 1 Condition GOSUB
INSTW	Establish Thumbwheel Data Inputs	ONVARB	On Numeric Variable 2 Condition GOSUB
INTHW	Enable Checking for Alarm Events	[OR]	Or [operator]

Command	Description		Command	Description
OUT	Activate Programmable Outputs		REGLOD	Registration Lockout Distance
[OUT]	Programmable Outputs Status [operator]		REGSS	Registration Single-Shot
OUTALL	Activate Programmable Outputs, Range		REPEAT	REPEAT Statement
OUTEN	Disable Programmable Outputs		RESET	Reset the 6K Controller
OUTFNC	Programmable Output Function Assignment		RUN	Begin Executing a Program
OUTLVL	Programmable Output Active Level		S	Stop Motion
OUTPn	Output on Position — Axis Specific		SCALE	Enable Scaling Factors
OUTPLC	Establish PLC Strobe Outputs		SCANP	Scan a Compiled PLC Program
OUTTW	Establish Thumbwheel Strobe Outputs		SCLA	Acceleration Scale Factor
PA	Path Acceleration		SCLD	Distance Scale Factor
PAA	Path Acceleration, S-curve		SCLMAS	Master Axis Scale Factor
PAB	Path Absolute		SCLV	Velocity Scale Factor
PAD	Path Deceleration		[SEG]	Number of Free Segment Buffers [operator]
PADA	Path Deceleration, S-curve		SFB	Select Servo Feedback Source
[PANI]	Position of ANI Inputs		SGAF	Gain – Acceleration Feedforward
PARCM	Radius-Specified CCW Arc Segment		SGENB	Enable a Servo Gain Set
PARCOM	Origin-Specified CCW Arc Segment		SGI	Gain – Integral Feedback
PARCOP	Origin-Specified CW Arc Segment		SGILIM	Gain – Integral Windup Limit
PARCP	Radius-Specified CW Arc Segment		SGP	Gain – Proportional Feedback
PAXES	Participating Axes for Contouring		SGSET	Save a Servo Gain Set
[PC]	Position Commanded [operator]		SGV	Gain – Velocity Feedback
[PCC]	Captured Commanded Position [operator]		SGVF	Gain – Velocity Feedforward
[PCE]	Position of Captured Encoder [operator]		[SIN()]	Sine [operator]
[PCME]	Position of Captured Master Encoder [operator]		SINAMP	Virtual Master Sine Wave Amplitude
[PCMS]	Position of Captured Master Cycle [operator]		SINANG	Virtual Master Sine Wave Angle
PCOMP	Compile a Profile or Program		SINGO	Virtual Master - Start Internal Sine Wave
[PE]	Position of Encoder [operator]		SMPER	Maximum Allowable Position Error
[PER]	Position Error [operator]		SOFFS	Servo Control Signal Offset
PESET	Set Encoder Absolute Position (steppers)		[SQRT]	Square Root [operator]
PEXE	Execute a Compiled Program		[SS]	System Status [operator]
[PI]	Pi (π) [operator]		STARTP	Start-up Program
PL	Select Path Local/Work Coordinate System		STEP	Enable Single Step Mode
PLC	Define Path Local Coordinates		STRGTD	Target Zone Distance
PLCP	Compiled PLC Program		STRGTE	Enable Target Zone Mode
PLIN	Move in a Line (Line Segment)		STRGTT	Target Zone Timeout Period
PLN	End of Loop, Compiled Motion		STRGTV	Target Zone Velocity
PLOOP	Start of Loop, Compiled Motion		[SWAP]	Task Swap Assignment [operator]
[PMAS]	Current Master Cycle Position [operator]		T	Time Delay
[PME]	Position of Master Encoder [operator]		[TAN()]	Tangent [operator]
PMECLR	Clear Master Encoder Absolute Position		TANI	Transfer ANI Analog Input Voltage
PMESET	Set Master Encoder Absolute Position		TAS	Transfer Axis Status
PORT	Designate Destination COM Port		TASF	Transfer Axis Status (full-text report)
POUT	Compiled Output (Contouring)		[TASK]	Task Number Assignment [operator]
POUTn	Compiled Output (Compiled Motion), Axis Specific		TASX	Transfer Axis Status, Extended
PPRO	Path Proportional Axis		TASXF	Transfer Axis Status, Extended (full-text)
PRTOL	Path Radius Tolerance		TCMDER	Transfer Command Error
PRUN	Run a Compiled Profile		TDAC	Transfer DAC Voltage
PS	Pause Program Execution		TDIR	Transfer Program Directory
PSET	Establish Absolute Position Reference		TDPTR	Transfer Data Pointer Status
[PSHF]	Net Position Shift Status [operator]		TER	Transfer Error Status
[PSLV]	Commanded Follower Position [operator]		TERF	Transfer Error Status (full-text report)
PTAN	Path Tangent Axis Resolution		TEX	Transfer Program Execution Status
PUCOMP	Un-Compile a Compiled Profile		TFB	Transfer Position of Feedback Devices
PULSE	Step Output Pulse Width		TFS	Transfer Following Status
PV	Path Velocity		TFSF	Transfer Following Status (full-test report)
PWC	Path Work Coordinates		TGAIN	Transfer Servo Gains
RADIAN	Specify Units in Radians or Degrees		[TIM]	Current Timer Value [operator]
RE	Enable Registration		TIMINT	Timer Value to Cause an Alarm Event
[READ]	Read a Value		TIMST	Start Timer
REG	Registration Distance		TIMSTP	Stop Timer

Command	Description	Command	Description
TIN	Transfer Programmable Input Status	TSS	Transfer System Status
TINO	Transfer Other Input Status	TSSF	Transfer System Status (full-text report)
TINOF	Transfer Other Input Status (full-text report)	TSTAT	Transfer Controller Statistics
TIO	Transfer Expansion I/O Status	TSTLT	Transfer Settling Time
TLABEL	Transfer Labels	TSWAP	Transfer Currently Active Tasks
TLIM	Transfer Hardware Limit Status	TTASK	Transfer Task Number
TMEM	Transfer Memory Usage	TTIM	Transfer Timer Value
TNMCY	Transfer Master Cycle Number	TTRIG	Transfer Trigger Interrupt Status
TNTMAC	Transfer Ethernet Address	TUS	Transfer User Status
TOUT	Transfer Programmable Output Status	TVEL	Transfer Current Commanded Velocity
TPANI	Transfer Position of ANI Inputs	TVELA	Transfer Current Actual Velocity
TPC	Transfer Commanded Position	TVMAS	Transfer Current Master Velocity
TPCC	Transfer Captured Commanded Position	[TW]	Thumbwheel Assignment [operator]
TPCE	Transfer Position of Captured Encoder	UNTIL()	Until Part of REPEAT Statement
TPCME	Transfer Position of Captured Master Encoder	[US]	User Status [operator]
TPCMS	Transfer Position of Captured Master Cycle	V	Velocity
TPE	Transfer Position of Encoder	[V]	Velocity [operator]
TPER	Transfer Position Error	VAR	Numeric Variable Assignment
TPMAS	Transfer Position of Master (current cycle)	VARB	Binary Variable Assignment
TPME	Transfer Position of Master Encoder	VARCLR	Clear All Variables
TPROG	Transfer Program Contents	VARI	Integer Variable Assignment
TPSHF	Transfer Net Position Shift	VARS	String Variable Assignment
TPSLV	Transfer Commanded Position of Follower	VCVT()	Variable Type Conversion
TRACE	Enable Program Trace Mode	[VEL]	Commanded Velocity Assignment [operator]
TRACEP	Enable Program Flow Mode	[VELA]	Actual Velocity Assignment [operator]
TRANS	Enable Translation Mode	VF	Final Velocity
TREV	Transfer Revision Level	[VMAS]	Velocity of Master [operator]
TRGFN	Trigger Functions	WAIT()	Wait for a Specific Condition
[TRIG]	Trigger Interrupt Status [operator]	WHILE()	WHILE Statement
TRGLOT	Trigger Interrupt Lockout Time	WRITE	Write a Message
TSCAN	Transfer Scan Time of PLC Program	WRVAR	Write a Numeric Variable
TSEG	Transfer Number of Free Segment Buffers	WRVARB	Write a Binary Variable
TSGSET	Transfer Servo Gain Set	WRVARI	Write a Integer Variable
TSKAX	Task Axis Association for Multi-Tasking	WRVARS	Write a String Variable
TSKTRN	Task Turns Before Swapping	XONOFF	Enable XON/XOFF ASCII Handshaking

Appendix B: ASCII Table

DEC	HEX	CHAR	DEC	HEX	CHAR	DEC	HEX	CHAR
0	00	NUL	42	2A	*	84	54	T
1	01	SOH	43	2B	+	85	55	U
2	02	STX	44	2C	,	86	56	V
3	03	EXT	45	2D	-	87	57	W
4	04	EOT	46	2E	.	88	58	X
5	05	ENQ	47	2F	/	89	59	Y
6	06	ACK	48	30	∅	90	5A	Z
7	07	BEL	49	31	1	91	5B	[
8	08	BS	50	32	2	92	5C	\
9	09	HT	51	33	3	93	5D	]
10	0A	LF	52	34	4	94	5E	^
11	0B	VT	53	35	5	95	5F	_
12	0C	FF	54	36	6	96	60	`
13	0D	CR	55	37	7	97	61	a
14	0E	SO	56	38	8	98	62	b
15	0F	S1	57	39	9	99	63	c
16	10	DLE	58	3A	:	100	64	d
17	11	XON	59	3B	;	101	65	e
18	12	DC2	60	3C	<	102	66	f
19	13	XOFF	61	3D	=	103	67	g
20	14	DC4	62	3E	>	100	68	h
21	15	NAK	63	3F	?	105	69	i
22	16	SYN	64	40	@	106	6A	j
23	17	ETB	65	41	A	107	6B	k
24	18	CAN	66	42	B	108	6C	l
25	19	EM	67	43	C	109	6D	m
26	1A	SUB	68	44	D	110	6E	n
27	1B	ESC	69	45	E	111	6F	o
28	1C	FS	70	46	F	112	70	p
29	1D	GS	71	47	G	113	71	q
30	1E	RSt	72	48	H	114	72	r
31	1F	US	73	49	I	115	73	s
32	20	SPACE	74	4A	J	116	74	t
33	21	!	75	4B	K	117	75	u
34	22	"	76	4C	L	118	76	v
35	23	#	77	4D	M	119	77	w
36	24	$	78	4E	N	120	78	x
37	25	%	79	4F	O	121	79	y
38	26	&	80	50	P	122	7A	z
39	27	`	81	51	Q	123	7B	{
40	28	(	82	52	R	124	7C	|
41	29	)	83	53	S	125	7D	}

DEC	HEX	CHAR
126	7E	~
127	7F	DEL
128	80	Ç
129	81	ü
130	82	é
131	83	â
132	84	ä
133	85	à
134	86	å
135	87	ç
136	88	ê
137	89	ë
138	8A	è
139	8B	ï
140	8C	î
141	8D	ì
142	8E	Ä
143	8F	Å
144	90	É
145	91	æ
146	92	Æ
147	93	ô
148	94	î
149	95	ò
150	96	û
151	97	ù
152	98	ÿ
153	99	Ö
154	9A	Ü
155	9B	¢
156	9C	£
157	9D	¥
158	9E	Pt
159	9F	f
160	A0	á
161	A1	í
162	A2	ó
163	A3	ú
164	A4	ñ
165	A5	Ñ
166	A6	ª
167	A7	º
168	A8	¿
169	A9	⌐
170	AA	¬
171	AB	½

DEC	HEX	CHAR
172	AC	¼
173	AD	¡
174	AE	«
175	AF	»
176	B0	░
177	B1	▒
178	B2	▓
179	B3	│
180	B4	┤
181	B5	╡
182	B6	╢
183	B7	╖
184	B8	╕
185	B9	╣
186	BA	║
187	BB	╗
188	BC	╝
189	BD	╜
190	BE	╛
191	BF	┐
192	C0	└
193	C1	┴
194	C2	┬
195	C3	├
196	C4	─
197	C5	┼
198	C6	╞
199	C7	╟
200	C8	╚
201	C9	╔
202	CA	╩
203	CB	╦
204	CC	╠
205	CD	═
206	CE	╬
207	CF	╧
208	D0	╨
209	D1	╤
210	D2	╥
211	D3	╙

DEC	HEX	CHAR
212	D4	╘
213	D5	╒
214	D6	╓
215	D7	╫
216	D8	╪
217	D9	┘
218	DA	┌
219	DB	█
220	DC	▄
221	DD	▌
222	DE	▐
223	DF	▀
224	E0	α
225	E1	β
226	E2	Γ
227	E3	π
228	E4	Σ
229	E5	σ
230	E6	∝
231	E7	τ
232	E8	Φ
233	E9	θ
234	EA	Ω
235	EB	δ
236	EC	∞
237	ED	∅
238	EE	∈
239	EF	∩
240	F0	≡
241	F1	±
242	F2	≥
243	F3	≤
244	F4	⌠
245	F5	⌡
246	F6	÷
247	F7	≈
248	F8	°
249	F9	·
250	FA	·
251	FB	√
252	FC	η
253	FD	2
254	FE	●
255	FF	

Appendix C: 6K vs. 6000 Programming Differences

The 6K Series product family is a "next generation" version of the existing 6000 Series product family. Because of changes in hardware and enhancements to firmware features, the command language (now referred to as the "6K command language") is slightly different.

Features That Work Differently

I/O Handling

- Programmable and analog I/O configuration is different. Referencing requires brick identifier (<B>) — a "brick" is onboard I/O or an extended I/O brick that has any combination of 8 SIM modules for digital and analog I/O. I/O bit patterns no longer conform to old syntax. Onboard I/O are considered I/O brick zero (0). New status command (TIO) displays the controller's I/O configuration. The "@" prefix makes a command apply to all I/O bricks (e.g., @TIN reports status of all inputs on all I/O bricks).

- Position capture only captures the dedicated axis (not all axes anymore); ANI position can no longer be captured. When trigger nA or nB is activated, the 6K performs a "hardware capture" (encoder/commanded position capture accuracy is ± 1 count) of the dedicated "n" axis: Servos capture the encoder and commanded; steppers capture the commanded position by default, or the encoder position if encoder capture is enabled (ENCCNT1). If the axis is a slave in a Following application, the position of the associated master is also captured (interpolated, accuracy is 50 µs multiplied by the velocity at the time of the capture). The Master Trigger ("TRIG-M") does a hardware capture of the "Master Encoder and all axes (encoder on servo axes, commanded (if ENCCNT0) or encoder (if ENCCNT1) on stepper axes). TSS/SS bits 25-28 no longer indicate position captures; instead, this status information is now reported with TTRIG/TRIG.

- OUT, OUTALL, and POUTn no longer reference only the programmable outputs that have the default function assignment (OUTFNCi-A); instead, OUT, OUTALL, and POUTn now reference all programmable outputs by their absolute reference as onboard outputs or outputs on a serial I/O brick. An attempt to change an output that is not an OUTFNCi-A output will elicit an error message (message is TBD), and the command will be ignored, but command processing will keep going. No error bit is set.

- Joystick control method uses digital inputs (external serial I/O, or onboard limits or triggers) and INFNC or LIMFNC definitions (M = Release, N = Axis select, and O = Velocity select). JOYAXL, JOYAXH, JOYCDB, JOYCTR, JOYEDB, and JOYZ syntax is changed to accommodate (e.g., JOYAXH3-2 assigns analog input #2 on brick #3 to control axis 1). Range for the analog channels is -10 to +10VDC (on existing products, it's 0-2.5VDC). TINOF (bits 1-5) no longer reports joystick input status—instead, refer to the respective TIN or TLIM bits for the inputs configured as joystick inputs (M, N, and O function assignments). Because the analog input voltage range is increased to +/- 10V with 12-bit resolution and because of the requirement to use external I/O, the syntax, ranges and defaults for joystick setup commands are changed (JOYCDB, JOYCTR, JOYEDB).

- INPLC, OUTPLC, INSTW, & OUTTW now require the I/O brick prefix (<B>); otherwise, these commands are applied only to the on-board inputs or outputs. All thumbwheel inputs must be on the same brick.

- INDUST syntax is altered. To use the system status (selection "I"), you should prefix the INDUST command with the task # for the system status you want (otherwise the default is task 1) – e.g., 3%INDUST16-2I assigns user status bit 16 to the 2nd bit of system status for task 3. To use the IN input status (selection "J"), you must prefix the INDUST command with the I/O brick # (otherwise, the default is on-board triggers) – e.g., 2INDUST14-4J assigns user status bit 14 to the status of I/O point 4 on I/O brick 2 (2IN.4).

- Debounce: INDEB is now brick-specific (all triggers constitute brick 0) — i.e., the INDEB value applies to all inputs on the specified brick. Therefore the syntax is changed to <!><B>INDEB<i> (drop the first integer field, which was used for the input number). INDEB now works the same for all general-purpose and trigger inputs (using the functionality the was previously applicable only to the general-purpose inputs). A new command, TRGLOT, sets the lock-out time for only the triggers defined as Position Capture inputs (INFNCi-H) — TRGLOT overrides the INDEB setting for the affected trigger inputs. INDEB also applies to limit inputs that are assigned standard input functions (e.g., Stop input) with the LIMFNC command. If an input is assigned a limit input function (R, S, or T), the input is not debounced (INDEB has no effect).

- Limit functions are added to INFNC so that external inputs can be used as limit inputs: R = positive limit,

S = negative limit, T = home limit. If an input is assigned a limit input function, it is no longer debounced (INDEB has no affect on it), and LH applies to the input, according to the axis and limit function assignment. It is still affected by INEN.

- The new LIMEN command adds INEN functionality for physical limit inputs (on the "LIMITS" connector), regardless of the inputs' assigned LIMFNC function. LH is still used to enable checking the state of the EOT limits (LH1, LH2 or LH3 is still required to detect errors).

- Limit inputs are programmable with the LIMFNC command. The default power-up state is such that each limit input is assigned to the correct LIMFNC function (e.g., the positive travel EOT limit for axis one is assigned LIMFNC1-1R). When an external input on and I/O brick is assigned a limit function, the user should reassign the hardware limit input as a general-purpose input (LIMFNCi-A) or as a different non-limit LIMFNC function. When left in the default function assignment (R, S, or T), the limits are not debounced; but if assigned other LIMFNC functions (including LIMFNCi-A), they are debounced with the INDEB setting for the on-board inputs.

- The functions of LHLVL and HOMLVL have been consolidated into LIMLVL (same bit assignments as LIM & LIMEN).

- INFEN and OUTFEN have been removed. Use the new DRFEN command to enable or disable checking the drive fault input.

- The digital outputs on the serial I/O bricks will be sinking or sourcing, depending on the jumper setting (on the I/O brick). The controller will auto-detect the jumper setting on power up and set the OUTLVL accordingly.

- TRGFN syntax changes: It now requires an axis identifier prefix, eliminating the succession of 8-bit patterns for each axis. Only A, B and M are allowed for the trigger identifier. When web registration (Following enhancements) is released, an additional function will be added to the TRGFN syntax.

- Analog input voltage range can be set with ANIRNG. Default is -10V to +10V. Other options are: 0 to +5V, -5 to +5V, and 0 to +10V.

Stepper Axis Behavior

- No Encoder position feedback (closed loop) features for steppers. MANY references to operation being different based on ENC. These features are no longer available to steppers: ENC Mode, Position Maint. features, Target Zone mode, and TPER & PER.

- Can't capture encoder position unless ENCCNT1 is used (and then commanded position can't be captured).

- Steppers no longer stop instantaneously on a kill, drive fault, limit, etc. — steppers now stop at LHAD/LHADA.

- Stepper axes now support s-curve profiles.

- Commanded position (AKA "motor" position) for steppers is now reported with the PC, TPC, & TPCC commands. The PM, TPM, PCM, and TPCM commands can still be in the product but they will not be documented.

- FOLK is now applicable to steppers.

- Output on Position works for steppers, now.

Encoders

- EFAIL added to detect encoder failures. Error reported with TASX bit 5 and error bit 17. Works only on differential encoders. By default, the 6K is compatible with differential encoders, but if you jumper pins 8 & 9 on the encoder connector (this feature requires a PCB modification) you can connect a single-ended encoder.

- New commands for addressing only the "Master Encoder", to check position (TPME, PME), captured position (TPCME, PCME), set absolute position (PMESET, PMECLR), change polarity (MEPOL), change to step and direction input (MESND).

- "FOLSND" is changed to "ENCSND" to avoid confusion about the functionality (does not depend on Following). FOLSND is still available as a hidden command for users of existing 6000 products.

- No counter commands (CNTE, CNT, CNTINT, CNTR). Instead, you must use ENCSND.

New Error Messages

- "ALTERNATIVE TASK NOT ALLOWED": Attempted to execute a LOCK command in another task.

- "AXIS NOT PART OF TASK": A task is attempting to execute a contouring path whose participating axis or axes (PAXES) are not associated with the task (TSKAX).

- "COMMAND/DRIVE MISMATCH": The command (or ≥ one field in the command) is not appropriate to the AXSDEF configuration (e.g., attempting to execute a servo tuning command on a stepper axis).

- "COMMAND NOT ALLOWED IN PROGRAM": Attempted to place a non-allowed command (e.g., scaling command) in a program.

- "INCORRECT BRICK NUMBER": Attempted to execute a command that addresses an I/O brick that is not connected to your 6K controller.

- "INVALID TASK IDENTIFIER": Attempting to launch a PEXE or EXE command into the supervisor task (task 0).

- "INPUT NOT DEFINED AS JOYSTICK INPUT": Attempting to execute JOYCDB, JOYCTR, JOYEDB, or JOYZ before executing JOYAXH or JOYAXL to assign the analog input to an axis.

Communications

- COM1 is the connector labeled "RS-232" or "ETHERNET", and COM2 is the connector labeled "RS-232/485".

- LOCK allows users to tie up the COM ports for a specific task (affects port handling — [,], DRPCHK, PORT)

- RS-422 in 6K only

- Baud rate adjusted with BAUD command (new) — default is 9600.

- Fast status (FASTAT) is removed. Status information is fixed and accessed through the Communications Server.

- Interrupts are now "alarms" and are available only through use of the Communications Server and ActiveX control (via Ethernet only)

- RP240 canned menus are different (run only); DVARI & DVARB are new; DSTP to enable/disable RP240 Stop button.

Miscellaneous

- Scaling changes: Scaling commands are now automatically stored in battery-backed RAM, and they are no longer allowed in a program (must be outside — this is handled by Motion Planner). Separate contouring and linear interpolation path scaling parameters (PSCLA, PSCLD, PSCLV) are no longer required; instead, use the SCLD value to scale all path motion (accel/decel, velocity, and distance).

- Command syntax & reporting formats have been modified to accommodate 8 axes (e.g., added fields per axis).

- Servo updates are fixed (this obsoletes the SSFR & INDAX commands); Only one system update now (2 ms)

- Status bit information (axis, system, error, user, Following, TSTAT, etc.) — deletions, additions, alterations:

  ASX: bit 5 for encoder fail,
      bit 6 for Z channel state (1 = active, 0 = inactive)
  ER: bit 16 for command error (Cleared with TCMDER),
      bit 17 for encoder fail (if EFAIL1. Cleared with EFAIL0)

- Error conditions/handling — Each task has its own error status register and error program.

- SYNTAX: Syntax change for REG: axis specific, only A and B are allowed.

- SYNTAX: Bit select syntax (syntax that required "-" before now allows you to use "=" instead, but not vice versa)

- SYNTAX: New ability to address a command to a specific address (and group of addresses of n, n+1, n+2, etc.)

- Timer enhancement: TIMST syntax has an additional optional field (<r>) — new syntax is TIMST<b>,<r>. If TIMST0, then <r> represents an absolute time; if TIMST1, then <r> represents a task number (timer will resume with the value of the timer for the specified task). Timer resolution is fixed at 2 ms.

- Program interrupts (ON conditions) — Each task has its own ON conditions and ONP program.

- Memory allocation for all 6K products most closely resembles the allocation for existing AT6000 products.

- 6K does not support COMEXK and COMEXP modes.

- Following enhancements— Geared Advance (FGADV) and virtual master and sine wave (FVMACC, FVMFRQ, SINAMP, SINANG, SINGO).

- Contouring enhancements — axes may now have different DRES and PULSE (steppers) and feedback resolutions (servo). Mechanical resolutions may also be different.

New Commands/Features for the 6K

Commands

Command	Description
%	Task Identifier
ANIEN	Analog Input Enable
ANIFB	Assign Analog Inputs as Axis Feedback
ANIMAS	Assign Analog Inputs to Axes
ANIRNG	Analog Input Voltage Range
AXSDEF	Axis Definition
BAUD	Baud Rate
[DKEY]	Value of RP240 Key
DRFEN	Drive Fault Input Enable
DSTP	Enable/Disable RP240 Stop Key
DVARB	Display Binary Variable on RP240
DVARI	Display Integer Variable on RP240
EFAIL	Encoder Failure Detect
ENCCNT	Encoder Count Reference Enable
ENCSND	Encoder Step & Direction Mode
EXE	Execute Program from a Compiled PLCP Program
FGADV	Following Geared Advance
FVMACC	Virtual Master Count Frequency Acceleration
FVMFRQ	Virtual Master Count Frequency
LIMEN	Enable Limit Inputs Defined as Non-Limit Inputs
LIMFNC	Limit Input Function
LIMLVL	Limit Input Active Level
LOCK	Lock Resource to Task
MEPOL	Master Encoder Polarity
MESND	Master Encoder Step & Direction Mode
NTADDR	IP Address for Ethernet Communication
NTMASK	Network Mask for Ethernet Communication
[PCMS]	Captured Master Cycle Position
PESET	Establish Encoder Absolute Position Reference
PEXE	Execute Compiled Prog. from Compiled PLCP Prog.
PLCP	Compiled PLC Program
[PCME]	Captured Master Encoder Position
[PME]	Position of Master Encoder
PMECLR	Clear Master Encoder Absolute Position Reference
PMESET	Establish Master Encoder Absolute Position Reference

SCANP ... Scan Compiled PLCP Program
SINAMP .. Virtual Master Internal Sine Wave
         Amplitude
SINANG .. Virtual Master Internal Sine Wave Angle
SINGO .... Virtual Master - Initiate Internal Sine Wave
[SWAP] . Task Swap Assignment
[TASK] .. Task Number Assignment
TIO .... Transfer Current Expansion I/O
         Configuration
TNTMAC . Transfer Ethernet Address
TPCME .... Transfer Position of Captured Master
         Encoder
TPCMS ... Transfer Captured Master Cycle Position
TPME....... Transfer Position of Master Encoder
TRACEP . Program Flow Mode Enable
TRGLOT . Trigger Lockout Time
[TRIG] ... Trigger Capture Status
TSCAN..... Scan Time of Last PLCP Program
TSKAX .... Task Axis
TSKTRN .. Task Turns Before Swapping
TSWAP ... Transfer Task Swap
TTASK .... Transfer Task Number
TTRIG ..... Transfer Trigger Capture Status
VARI .... Integer Variable Assignment
WRVARI . Write an Integer Variable

Features

- Multi-tasking (impact: task-specific commands & report-backs, and syntax)

- PLC Scan mode

- Following enhancements: Geared Advance and Virtual Master

- Integer variables (VARI)

- Master encoder handling

- TRACEP (trace mode enhancement)

- Compiled conditionals (not first release)

- Baud rate changed with command (BAUD) only

- RP240 menus, DVARI, DVARB, DSTP

- Ethernet communication and "alarm" event handling

- Additional syntax symbols (% for addressing specific tasks, <B> for addressing specific I/O bricks)

6000 Commands not in the 6K Command Language

ANI Option
    ANIPOL
    ANV
    ANVO
    ANVOEN
    CA
    PCA
    TCA
    TPCA

Counter
    CNT
    CNTE
    CNTINT
    CNTR
    TCNT

Command Processing
    COMEXK
    COMEXP

Encoder
    EMOVDB
    ENC
    EPM
    EPMDB
    EPMG
    EPMV

Feedrate
    FR
    FRA
    FRH
    FRL
    FRPER

Servo (misc.)
    INDAX
    SSFR
    SDTAMP
    SDTFR

Interrupt
    INTCLR
    TINT

Data Streaming Mode
    SD
    STD
    STREAM

Scaling (contouring and linear interpolation only)
    PSCLA (use SCLA instead)
    PSCLD (use SCLD instead)
    PSCLV (use SCLV instead)

Product Specific: ZETA610n
    DACTDP
    DAREN
    DAUTOS
    DELVIS
    DMTIND
    DMTSTT
    DWAVEF

Product Specific: APEX615n
    DRESET

Product Specific: 6270
    DACMIN
    LDT
    LDTGRD
    LDTPOL
    LDTRES
    LDTUPD
    PCL
    SGAFN
    SGIN
    SGPN
    SGVFN
    SGVN
    SOFFSN
    SSWD
    SSWG
    TLDT
    TPCL

Miscellaneous
    OUTANA
    SSV
    TANV
    TEST
    FASTAT

Index

commanded acceleration,
feedforward gain, 233
commanded direction polarity, 50
commanded position, 196, 271
capture, 128, 190, 267
comparison or assignment, 189
display, 266
follower
assignment/comparison, 210
transfer, 273
commands
buffered, 22
looping, 158
looping, compiled, 201
command buffer execution
after end-of-travel limit
(COMEXL), 52
after pause/continue input
(COMEXR), 52
after stop (COMEXS), 53
continuous (COMEXC), 50
command description format, 2
command field symbols, 3
command list, 303
command value substitutions, 5
command-to-product
compatibility, 2
default settings, 2
delimiters, 4
immediate, 22
syntax, 2
types, 2
comment delimiter, 4, 22
communication interface
addressing units in a chain, 39
baud rate, 47
COM port selection (PORT), 204
controlling multiple COM ports,
204
send response to alternate port,
34
send response to both ports, 34
echo enable, 75
enable communication (E), 75
Ethernet
Ethernet address, 265
IP address, 170
network mask, 170
lock COM port, 158
RP240 check, 71
XON/XOFF, enable & disable,
302
communications server, 135, 136
compiled motion. See PLC program
compiling (PCOMP), 194
execute from PLC program
(PEXE), 197
failed PCOMP, 242, 281
final velocity, 298
GOBUF segments, 108
looping, 201
memory status, 242, 281
outputs (POUTn command), 205
run the profile (PRUN), 208
status of program storage, 253
status, free segments, 231, 278
uncompile the profile
(PUCOMP), 212
compiling a profile or program
(PCOMP), 194
conditional branching

ELSE, 76
IF, 122
NIF, 168
NWHILE, 170
REPEAT, 221
UNTIL, 289
WHILE, 299
conditional go, 112
continue (!C), 47, 49, 53, 150
continuous positioning mode (MC1),
164
continuous shift. See shift,
continuous
contouring, 183
axes, inclusion of, 189
memory allocation, 165
path
absolute (PAB), 184
acceleration (PA), 183
acceleration, s-curve, 183
CCW arc, origin specified,
187
CCW arc, radius specified,
186
compile (PCOMP), 194
compile (PCOMP), failure,
242, 281
CW arc, origin specified, 188
CW arc, radius specified, 188
deceleration (PAD), 184
deceleration, s-curve, 185
definition (DEF/END), 63, 78
execute from PLC program
(PEXE), 197
incremental (PAB), 184
line segment definition, 200
local coordinates (PLC), 199
local mode (PL), 198
memory allocation, 165
memory status, 253
outputs (POUT), 205
proportional axis (PPRO), 206
radius tolerance (PRTOL), 207
run/execute (PRUN), 208
s-curve accel/decel, 183, 185
tangent axis resolution
(PTAN), 211
uncompile (PUCOMP), 212
velocity (PV), 213
work coordinates (PWC), 213
scaling (SCLD), 18
control characters, 300
control signal offset, 241
controlling multiple serial ports
select target port (PORT), 204
send response to alternate port, 34
send response to both ports, 34
coordinates, contouring
absolute, 184
incremental, 184
cosine, 54, 214

D

DAC
limit, 57
value
assignment/comparison, 56
status, 253
damping, 237
data

assignment (DAT), 57
fields, in command syntax, 3
program (DATP), 57, 58
program size (DATSIZ), 60
read from the RP240, 67
statement (DATA), 57
memory required, 165
storage, 57, 58, 60, 254
teach, 61
transfer, 181, 182
deadband
joystick center, 145
joystick end, 145
stall, 88
debounce time for programmable
inputs, 124
debounce time for trigger interrupt
inputs, 277
debugging tools. See Programmer's
Guide and back cover
axis status (extended) report, 250
axis status report, 248
break point, 47
break, manual, 48
ENABLE status, 261, 262
error messages, 9
HALT, 115
identify bad command, 12, 252
single-step mode, 24, 243
technical support, 115
trace mode, 273, 274
translation mode, 275
deceleration, 37
assignment/comparison, 38
change on-the-fly, 51
limits
hard, 151
soft, 160
path, 184
scaling, 17
s-curve profiling, 13, 38
hard limits, 151
homing, 118
jogging, 139
joystick, 143
paths, 185
soft limits, 160
default command settings, 2
define
program/subroutine/path (DEF),
62
user status, 125
degrees, unit of measure, 214
delay time (T command), 247
delete a program/subroutine/path
(DEL), 63
delimiter, comment, 4, 22
delimiters, command, 4
digital-to-analog converter (DAC)
voltage, 56, 57, 253
direction polarity, commanded, 50
disable drive, 70
disable drive on kill, 149
display
messages, 25
RP240. See RP240
distance, 55
assignment, 56
change on-the-fly, 51, 55
fractional step truncation, 19,
228, 229

6K Series Commands — *Functional Grouping*

Alarm Events
INTHW
INTSW
TIMINT

Assignment & Comparison
A
AD
ANI
AS
ASX
D
DAC
DAT
DPTR
DREAD
DREADF
ER
FB
FS
IN
INO
LIM
MOV
NMCY
OUT
PANI
PC
PCC
PCE
PCMS
PE
PER
PMAS
PSHF
PSLV
READ
SEG
SS
SWAP
TIM
TRIG
TW
US
V
VAR
VARB
VARI
VARS
VEL
VELA
VMAS

Branching – Conditional
ELSE
IF
NIF
NWHILE
REPEAT
UNTIL
WHILE

Branching – Unconditional
GOSUB
GOTO
JUMP
L
LN
LX
PLN
PLOOP

Command Buffer Control
COMEXC
COMEXL
COMEXR
COMEXS
GOWHEN

Communication Interface
[
]
ADDR
BAUD
BOT
E
ECHO
EOL
EOT
ERRBAD

ERRDEF
ERRLVL
ERROK
LOCK
NTADDR
NTMASK
PORT
READ
RESET
TNTMAC
WRITE
WRVAR
WRVARB
WRVARI
WRVARS
XONOFF

Contouring
DEF
END
MEMORY
PA
PAA
PAB
PAD
PADA
PARCM
PARCOM
PARCOP
PARCP
PAXES
PCOMP
PEXE
PL
PLC
PLIN
POUT
PPRO
PRTOL
PRUN
PTAN
PUCOMP
PV
PWC
TDIR
TMEM

Controller Configuration
AXSDEF
CMDDIR
DRFEN
EFAIL
ENCCNT
INDUSE
INDUST
KDRIVE
MEMORY
NTADDR
PULSE
SFB
TIO
TNTMAC
TREV
TSTAT

Data Storage
DAT
DATA
DATP
DATPTR
DATRST
DATSIZ
DATTCH
DPTR
TDPTR
TW

Display (status)
TANI
TAS
TASF
TASX
TASXF
TCMDER
TDAC
TDIR
TDPTR
TER
TERF
TEX
TFB
TFS
TFSF

TGAIN
TIN
TINO
TINOF
TIO
TLABEL
TLIM
TMEM
TNMCY
TNTMAC
TOUT
TPANI
TPC
TPCC
TPCE
TPCME
TPCMS
TPE
TPER
TPMAS
TPSHF
TPSLV
TREV
TSCAN
TSEG
TSS
TSSF
TSTAT
TSTLT
TSWAP
TTASK
TTIM
TTRIG
TUS
TVEL
TVELA
TVMAS

Drive Config.
DRES
DRFEN
DRFLVL
DRIVE
KDRIVE
TMEM

Encoder
EFAIL
ENCCNT
ENCPOL
ENCSND
ERES
ESDB
ESK
ESTALL
FB
MEPOL
MESND
PCE
PCME
PE
PER
PESET
PME
PMECLR
SFB
TFB
TPCE
TPCME
TPE
TPER
TPME

Error Handling
ER
ERRBAD
ERRLVL
ERROR
ERRORP
TCMDER
TER
TERF

Following
ANIMAS
FFILT
FGADV
FMAXA
FMAXV
FMCLEN
FMCNEW
FMCP
FOLEN
FOLK
FOLMAS

FOLMD
FOLRD
FOLRN
FOLRNF
FPPEN
FSHFC
FSHFD
FVMACC
FVMFRQ
GOWHEN
NMCY
PCMS
PMAS
PSHF
PSLV
SCLD
SCLMAS
TFS
TFSF
TNMCY
TPCME
TPCMS
TPMAS
TPSHF
TPSLV
TRGFN
TVMAS
VMAS

Homing
HOM
HOMA
HOMAA
HOMAD
HOMADA
HOMBAC
HOMDF
HOMEDG
HOMV
HOMVF
HOMZ
IN
INFNC
LIM
LIMFNC
LIMLVL
TIN
TLIM

Inputs
ANI
ANIEN
ANIFB
ANIMAS
ANIRNG
DRFEN
DRFLVL
IN
INDEB
INEN
INFNC
INLVL
INO
INPLC
INSELP
INSTW
LIM
LIMEN
LIMFNC
LIMLVL
TIN
TINO
TINOF
TIO
TLIM
TRGFN
TRGLOT
TRIG
TTRIG

Interrupt to Program
ONCOND
ONIN
ONP
ONUS
ONVARA
ONVARB

Jogging
INFNC
JOG
JOGA
JOGAA
JOGAD

JOGADA
JOGVH
JOGVL
LIMFNC

Joystick
ANI
ANIEN
ANIRNG
INFNC
JOY
JOYA
JOYAA
JOYAD
JOYADA
JOYAXH
JOYAXL
JOYCDB
JOYCTR
JOYEDB
JOYVH
JOYVL
JOYZ
LIMFNC
TIO

Limits (End-Of-Travel)
INFNC
LH
LHAD
LHADA
LIM
LIMEN
LIMFNC
LIMLVL
LS
LSAD
LSADA
LSNEG
LSPOS
TLIM

Loops
L
LN
LX
PLN
PLOOP

Motion
A
[A]
AA
AD
[AD]
ADA
C
D
[D]
GO
GOWHEN
^K
K
MA
MC
MOV
PSET
PESET
S
V
[V]

Motion (Compiled)
FOLRNF
GOBUF
PCOMP
PEXE
PLN
PLOOP
PLN
PLOOP
POUT
PRUN
PUCOMP
SEG
TSEG
VF

Motion (Linear Interpolated)
D
GOL
PA

PAA
PAD
PADA
PV
SCLD

Motion (S-Curve)
AA
ADA
HOMAA
HOMADA
JOGAA
JOGADA
JOYAA
JOYADA
LHADA
LSADA
PAA
PADA

Multi-Tasking
% (Task I.D.)
LOCK
SWAP
TASK
TSKAX
TSKTRN
TSWAP
TTASK

Operators (Bitwise)
&
|
^
~
<<
>>

Operators (Logical)
AND
NOT
OR

Operators (Math)
=
( )
+
-
*
/
SQRT

Operators (Other)
!
@
;
$
#
'
. (period)
"
\
[
]

Operators (Relational)
=
>
>=
<
<=
<>

Operators (Trig)
ATAN
COS
PI
RADIAN
SIN
TAN

Outputs
OUT
[OUT]
OUTALL
OUTEN
OUTFNC
OUTLVL
OUTP
OUTPLC

PAA
PAD
PADA
PV
SCLD

PLC Program
EXE
PEXE
PLCP
SCANP
TSCAN

Power-Up Execution
RESET
STARTP

Program Debug Tools
#
ANIEN
BP
HELP
INEN
OUTEN
STEP
TCMDER
TRACE
TRACEP
TRANS

Program Definition & Execution
DATP
DEF
DEL
END
ERASE
ERRORP
EXE
INFNC
INSELP
MEMORY
ONP
PEXE
PLCP
RUN
SCANP
SEG
STARTP
TDIR
TEX
TMEM
TPROG
TSEG
$

Program Flow Control
BP
BREAK
C
ELSE
GOSUB
GOTO
GOWHEN
HALT
IF
JUMP
L
LN
LX
NIF
NWHILE
PS
REPEAT
T
TRACEP
UNTIL
WAIT
WHILE

Registration & Position Capture
INFNC
PCC
PCE
PCME
PCMS
RE
REG
REGLOD
REGSS

OUTTW
POUT
TIO
TOUT

PLC Program
EXE
PEXE
PLCP
SCANP
TSCAN

Power-Up Execution
RESET
STARTP

Program Debug Tools
#
ANIEN
BP
HELP
INEN
OUTEN
STEP
TCMDER
TRACE
TRACEP
TRANS

Program Definition & Execution
DATP
DEF
DEL
END
ERASE
ERRORP
EXE
INFNC
INSELP
MEMORY
ONP
PEXE
PLCP
RUN
SCANP
SEG
STARTP
TDIR
TEX
TMEM
TPROG
TSEG
$

Program Flow Control
BP
BREAK
C
ELSE
GOSUB
GOTO
GOWHEN
HALT
IF
JUMP
L
LN
LX
NIF
NWHILE
PS
REPEAT
T
TRACEP
UNTIL
WAIT
WHILE

Registration & Position Capture
INFNC
PCC
PCE
PCME
PCMS
RE
REG
REGLOD
REGSS

TPCC
TPCE
TPCME
TPCMS
TRGLOT
TRIG
TTRIG

RP240
DCLEAR
DJOG
DKEY
DLED
DPASS
DPCUR
DREAD
DREADF
DREADI
DRPCHK
DSTP
DVAR
DVARB
DVARI
DWRITE

Scaling
SCALE
SCLA
SCLD
SCLV
SCLMAS

Servo
CMDDIR
DAC
DACLIM
FB
SFB
SGAF
SGENB
SGI
SGILIM
SGP
SGSET
SGV
SGVF
SMPER
SOFFS
STRGTD
STRGTE
STRGTT
STRGTV
TDAC
TFB
TGAIN
TSGSET
TSTLT

Timer
TIM
TIMINT
TIMST
TIMSTP
TTIM

Variables
DVAR
DVARB
DVARI
DWRITE
VAR
VARB
VARCLR
VARI
VARS
VCVT
WRVAR
WRVARB