

CHAPTER THREE

Basic Operation Setup

IN THIS CHAPTER

This chapter will enable you to understand and implement these basic operation features:

- Before You Begin (setup programs, Motion Planner, resetting, etc.)..... 44
- Memory Allocation 45
- Drive Setup..... 46
- Scaling..... 48
- Positioning Modes..... 52
- End-of-Travel Limits..... 57
- Homing..... 59
- Encoder-Based Stepper Operation (stepper axes only) 64
- Servo Setup (servo axes only)..... 67
- Target Zone Mode (servo axes only)..... 74
- Programmable Inputs and Outputs (incl. triggers and auxiliary outputs)..... 75
- Variable Arrays (*teaching variable data*)..... 94

Before You Begin



WARNING



The 6K Product is used to control your system's electrical and mechanical components. Therefore, you should test your system for safety under all potential conditions. Failure to do so can result in damage to equipment and/or serious injury to personnel.

Setup Parameters Discussed in this Chapter

Below is a list of the setup parameters discussed in this chapter. You can check the status of each parameter setting by entering the respective setup command without any command fields (e.g., typing LIMFNC <cr> displays the current function and state of each limit input). Some setup parameters are also reported with the TSTAT and TASF status commands (these and other status commands are described on page 226).

Setup Parameter	Command	See Pg	Setup Parameter	Command	See Pg
Memory (status with TDIR & TMEM)	MEMORY.....	12 & 45	Programmable Input Functions		75
Drive Setup.....		46	Define input functions	INFNC & LIMFNC	
Drive type (servo/stepper) selection.....	AXSDEF		Input active level	INLVL & LIMLVL	
Drive fault input - enable.....	DRFEN		Input debounce	INDEB	
Drive fault input - active level	DRFLVL		Trigger interrupt - special functions	TRGFN	
Drive resolution (stepper only)	DRES		Trigger interrupt - lockout time.....	TRGLOT	
Step pulse width (stepper only)	PULSE		Virtual Inputs.....	IN	
Drive disable on kill (servo only).....	KDRIVE		Programmable Output Functions.....		75
Drive stall detection	DSTALL		Define output functions.....	OUTFNC	
Scaling.....		48	Output active level	OUTLVL	
Enable scaling factor	SCALE		End-of-travel limits.....		57
Acceleration scaling factor.....	SCLA		Limit function assignments	LIMFNC & INFNC	
Distance scaling factor	SCLD		Hardware – enabled	LH	
Velocity scaling factor.....	SCLV		Hardware – deceleration.....	LHAD	
Positioning Mode.....		52	Hardware – s-curve decel.....	LHADA	
Continuous or preset	MC		Hardware – active level of input.....	LIMLVL	
Preset: absolute or incremental.....	MA		Software – enabled	LS	
Encoder Based Stepper Operation (stepper only)		64	Software – deceleration	LSAD	
Encoder resolution.....	ERES		Software – s-curve decel	LSADA	
Encoder capture/counting enable.....	ENCNT		Software – negative direction limit	LSNEG	
Encoder polarity.....	ENCPOL		Software – positive direction limit	LSPOS	
Encoder failure detection.....	EFAIL		Homing		59
Commanded direction polarity.....	CMDDIR		Limit function assignments	LIMFNC & INFNC	
Stall detection	ESTALL		Acceleration	HOMA	
Kill when stall is detected	ESK		S-curve acceleration	HOMAA	
Stall deadband.....	ESDB		Deceleration.....	HOMAD	
Servo Setup (servo axes only)		67	S-curve deceleration.....	HOMADA	
Tuning parameters	(see page Error!		Backup to home.....	HOMBAC	
Bookmark not defined.)			Final approach direction	HOMDF	
Feedback source selection.....	SFB, ANIFB		Stopping edge of switch.....	HOMEDG	
Feedback source resolution	ERES, ANIRNG		Home switch active level	LIMLVL	
Encoder failure detection.....	EFAIL		Velocity	HOMV	
Maximum position error	SMPER		Velocity of final approach.....	HOMVF	
Encoder feedback polarity	ENCPOL		Home to Z channel input.....	HOMZ	
Commanded direction polarity.....	CMDDIR		Variable Arrays (<i>teaching</i> variable data)		94
DAC output limit, maximum	DACLIM		Initialize numeric variable for data	VAR or VARI	
Servo control signal offset	SOFFS		Define data program and program size	DATSIZ	
Target Zone end-of-move settling criteria (servo axes).....		74	Set data pointer & establish increment	DATPTR	
Target zone mode enable	STRGTE		Reset data pointer to specific location..	DATRST	
Target distance zone	STRGTD				
Target velocity zone	STRGTV				
Target settling timeout period	STRGTT				

Using a Setup Program

The features described in this chapter are configured with certain 6K Series commands, commonly referred to as “setup commands.” We recommend placing these commands (except MEMORY) into a special “setup program” that is executed to prepare the 6K Series product for subsequent controller operations. Further details about setup programming are provided in the *Creating and Executing a Setup Program* section, page 13.

USE THE SETUP WIZARD IN MOTION PLANNER

The easiest way to create your setup program is to use the Setup Wizard in Motion Planner’s editor.

Resetting the Controller

There are two primary ways to reset the 6K controller (listed below).

- Cycle power.
- Execute the RESET command.

When the controller is reset, most of the previously entered command parameters are returned to their original factory default values. All programs and variables, as well as certain command values, are retained in non-volatile memory (see page 33). If a start-up program is assigned with the STARTP command, resetting the controller will automatically execute that program. If you are using an RP240, the RESET function is available if you use the default menu system (see page 114).

Memory Allocation

For details about memory allocation, see *Storing Programs* on page 11.

CAUTION

Issuing a new MEMORY command (e.g., MEMORY200000, 100000) will erase all existing programs and compiled contouring path segments residing in the 6K product’s memory. To determine the status of memory allocation, use the TMEM command.

Do not place the MEMORY command in the program assigned as the startup (STARTP) program. Doing so would erase all programs and segments upon cycling power or issuing the RESET command.

Drive Setup

Drive Type Selection

6K products can control any combination of stepper and servo axes. To tell the 6K which type of drive you are using and a particular axis, use the `AXSDEF` command. Stepper drives receive their positioning information via step and direction signals. Servo drives receive their positioning commands via a ± 10 volt signal.

NOTE: If a drive is attached at the time you issue the `AXSDEF` command, the drive must be disabled (`DRIVE0`).

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.

✍ Make your drive type selection before configuring other parameters that are relative to the drive type.

AXSDEF0 — Stepper Only Commands:		AXSDEF1 — Servo Only Commands:			
DRES	FMAXA	ANIFB	DSTALL	KDRIVE	PER
ENCCNT	FMAXV	SFB	SGP	SGI	SGV
ESDB	PULSE	SGVF	SGAF	SGILIM	SOFFS
ESK		SMPER	SGSET	SGENB	STRGTD
ESTALL		STRGTE	STRGTT	STRGTV	TFB
		TGAIN	TPER	TSGSET	TSTLT

Drive Fault Input

Enable the Drive Fault Input

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. The drive fault input must be enabled (`DRFEN1`) before you can use these functions (remember that the default power-up state is disabled):

- `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 it 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 (`OUTFNC-F`) will turn on when a drive fault occurs or a user fault input (`INFNC-F` or `LIMFNC-F`) is activated.

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

Set the Drive Fault Input Active Level

The drive fault level (`DRFLVL`) should be set to “active high” or “active low” for each axis. The default setting is “active high” (`DRFLVL1`). The drive fault input schematic is shown in your 6K Series product installation guide. Use the table below as a guide.

Compumotor Product	Drive Fault Level
GEMINI, APEX, Dynaserv, LN, OEM Series, S, TQ, ZETA	Active High (<code>DRFLVL1</code>)
SV, BLH, L, LE, PDS, PK130	Active Low (<code>DRFLVL0</code>)

NOTE: If you are using a drive that does not have a drive fault output, set the drive fault level to active low (`DRFLVL0`).

NOTE

Once the drive fault level has been configured, you must enable the drive fault input with the `DRFEN1` command before the input is usable.

Checking Drive Fault Input Status (see table below): Axis status bit 14 (TASF, TAS or AS commands) indicates the drive fault input status, but only while the drive is enabled (DRIVE1) and the drive fault input is enabled (DRFEN1). If you need to monitor the drive fault input status regardless of the state of DRIVE and DRFEN, use the extended axis status bit 4 (TASXF, TASX, or ASX commands).

Drive Fault Level (DRFLVL)	Status of device driving the Fault input	Axis Status of Bit 14 (or) Extended Axis Status Bit 4
DRFLVL1 (active high)	OFF or not connected (not sinking current) ON (sinking current)	1 (drive fault has occurred) Ø
DRFLVLØ (active low)	OFF or not connected (not sinking current) ON (sinking current)	Ø 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.

Drive Stall Detection (stepper axes only)

The DSTALL command determines if the Stall Input on pin 4 of the DRIVE connector will be checked as Drive Stall indicator.

The state of the Stall Input can be monitored at all times with Extended Axis Status bit 7 (reported with TASX, TASF, and the ASX assignment/comparison operand); if left unconnected, the input is low, and status bit 7 will be set (reports a “1”). If this input is enabled as a drive stall indicator with the DSTALL1 command, a low input will be interpreted as a Drive Stall.

When a Drive Stall is detected, the 6K responds as follows: (this response is the same as that for Encoder Stall Detection, which is enabled with the ESTALL command)

- The stall is reported with Axis Status bit 12 (reported with TAS, TASF, and AS).
- If ERROR error-checking bit 1 is enabled (ERROR.1-1):
 - The stall is reported with Error Status bit 1 (reported with TER, TERF, and ER).
 - The 6K branches to the assigned ERRORP program.
- If the Kill-on-Stall feature is enabled (ESK1), the 6K immediately stops pulses from being sent to the affected axis.

Drive Resolution (stepper axes only)

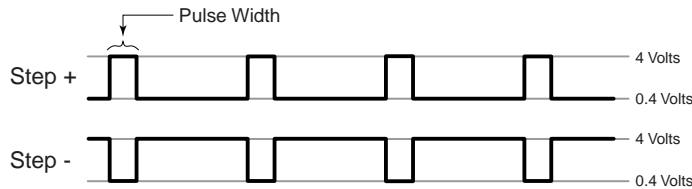
The drive resolution controls the number of steps the 6K controller considers as a full revolution for the motor drive. The controller's resolution is set with the DRES command (default is 4,000 steps/rev). Refer to the user documentation that accompanied your drive if you need to change its resolution.

IMPORTANT!

If the controller's resolution (set with the DRES command) does not match the drive's resolution, the motor will not move according to the programmed distance and velocity.

Step Pulse (stepper axes only)

The step output pulse width can be varied using the PULSE command. The pulse width can be 0.3 µs to 8 µs (default is 0.3 µs). The pulse width is the amount of time the step output signal is active (see illustration below). The step output pulse width should be configured to meet the minimum step input pulse width requirement of the motor drive you are using.



The pulse width does not vary as the motion profile is executed. The same pulse width is used during acceleration, constant velocity, and deceleration.

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 \rightarrow 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

Disable Drive On Kill (servo axes only)

Normally, when you issue a Kill command (K , $!K$, or $\langle\text{ctrl}\rangle K$) or activate an input configured as a kill input (see INFNCi-C or LIMFNCi-C command), motion is stopped at the hard limit (LHAD/LHADA) deceleration setting and the drives are left in the enabled state (DRIVE1).

However, your application can require you to *disable (shut down or de-energize)* the drives in a Kill situation to, for example, prevent damage to the motors or other system mechanical components. If so, set the controller to the *Disable Drive on Kill* mode with the KDRIVE1 command. In this mode, a kill command or kill input will shut down the drives immediately, letting the motors *free wheel* (without control from the drives) to a stop.

Scaling

Units of Measure without Scaling

Scaling is disabled ($\text{SCALE}\emptyset$) as the factory default condition:

- Stepper axes: When scaling is disabled, 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.

Motion Attribute	Units of Measure (per feedback source)	
	Encoder	ANI
Accel/Decel	Revs/sec/sec *	volts/sec/sec
Velocity	Revs/sec *	volts/sec
Distance	Counts (<i>steps</i>) **	Counts (<i>steps</i>) **

* All accel/decel & velocity values are multiplied by the ERES 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
Following	SCLA	SCLV	SCLD for follower distances SCLMAS for master distances
Contouring, Linear Interpolation	SCLD (used for all path motion scaling)		

When Should I Define Scaling Factors?

NOTE
All scaling parameters are saved in battery-backed RAM

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 after you cycle power or 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.

Servo Axes

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
```

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 for point-to-point motion (e.g., A, AA, AD, HOMA, HOMAD, JOGA, etc.) are multiplied by the SCLA command value. **NOTE:** All accel/decel commands for linear interpolated and contouring motion (e.g., PA, PAD, etc.) are multiplied by the SCLD command 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 SCLA10, the A9.9999 command would be truncated to A9.9).

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

Use the following equations to determine the range of acceleration and deceleration values for your product.

If the analog input voltage range is changed, the equations will change. See ANIRNG command for details.

Axis Type	Minimum Accel or Decel (resolution)	Maximum Accel or Decel
Stepper	$\frac{0.001 * DRES}{SCLA}$	$\frac{999.9999 * DRES}{SCLA}$
Servo	Encoder feedback: $\frac{0.001 * ERES}{SCLA}$	Encoder feedback: $\frac{999.9999 * ERES}{SCLA}$
	ANI feedback: $\frac{0.205}{SCLA}$	ANI feedback: $\frac{204799.9795}{SCLA}$

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 for point-to-point motion (e.g., V, HOMV, HOMVF, JOGVH, JOGVL, etc.) are multiplied by the SCLV command value. **NOTE:** The velocity commands for linear interpolated and contouring motion (PV and PVF) are multiplied by the SCLD command 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.

Maximum Velocity for Stepper Axes		Maximum Velocity for Servo Axes (determined by feedback source selected for axis 1)	
$\frac{6,500,000}{\text{SCLV}}$	$n = \text{maximum velocity as set by the PULSE command.}$	Encoder Feedback: $\frac{6,500,000}{\text{SCLV}}$	Equation changes if ANI voltage range is changed (see ANIRNG command).
		ANI Feedback: $\frac{1000 * 205}{\text{SCLV}}$	

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 for point-to-point motion (e.g., D, PSET, REG, SMPER) are multiplied by the SCLD command value. **NOTE:** Distance, accel/decel, and velocity commands for contouring and linear interpolated motion (e.g., PARCM, PARCOM, PARCOP, PARCP, PLC, PLIN, PRTOL, PWC, PV, PVF, PA, PAD, PAA, PADA) are also multiplied by the SCLD command value.

Scaling for Following Motion: The SCLD command defines the follower's 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: <i>Master Cycle Length</i>	FOLRN: <i>Follower-to-Master Ratio (Numerator)</i>
FMCP: <i>Master Cycle Position Offset</i>	FSHFD: <i>Preset Phase Shift</i>
FOLMD: <i>Master Distance</i>	GOWHEN: <i>Conditional GO (left-hand variable ≠ PMAS)</i>
FOLRD: <i>Follower-to-Master Ratio (Denominator)</i>	TPSHF & [PSHF]: <i>Net Position Shift of Follower</i>
GOWHEN: <i>Conditional GO (left-hand variable is PMAS)</i>	TPSLV & [PSLV]: <i>Position of Follower Axis</i>
TPMAS & [PMAS]: <i>Position of Master Axis</i>	
TVMAS & [VMAS]: <i>Velocity of Master Axis</i>	

Scaling for Contouring Motion: All distance, accel/decel, and velocity is scaled by the SCLD value.

Fractional Step Truncation
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.

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 SCLD400, 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

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 entering distance in inches (both axes)
SCLV125000,125000 ; Allow entering velocity in inches/sec (both axes)
SCLA125000,125000 ; Allow entering accel/decel in inches/sec/sec
```

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         ; Set encoder resolution to 4000 steps/rev (post quadrature)
SCALE1           ; Enable scaling
SCLD20000,40000  ; Allow entering distance values in inches
SCLV20000,40000  ; Allow entering velocity values in inches/sec
SCLA20000,40000  ; Allow entering accel/decel values in inches/sec/sec
SFB2             ; Select ANI feedback for axis 1
SCALE1           ; Enable scaling
SCLD205          ; Allow entering distance values in volts
SCLV205          ; Allow entering velocity values in volts/sec
SCLA205          ; Allow entering 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 and 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
PABO            ; 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
```

Positioning Modes

But first, a word about basic motion...

Accel, Decel,
Velocity, Distance

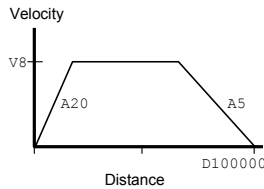
The basic motion profile comprises acceleration, deceleration, velocity, and distance commands (see table below). Motion is generally initiated with the GO command.

Parameter	Units (Unscaled), Stepper	Units (Unscaled), Servo	Unit Scaling Command *
Acceleration	revs/sec ²	encoder/resolver: revs/sec ² ANI: volts/sec ²	SCLA (or SCLD **)
Deceleration	revs/sec ²	encoder/resolver: revs/sec ² ANI: volts/sec ²	SCLA (or SCLD **)
Velocity	revs/sec	encoder/resolver: revs/sec ANI: volts/sec	SCLV (or SCLD **)
Distance	steps	counts	SCLD ***

* Scaling must first be enabled with the SCALE1 command. For details on scaling, refer to page 48.
 ** All Contouring and linear interpolated motion is scaled with SCLD (distance, accel/decel, and velocity).
 *** An axis assigned as a master (for Following) is scaled by the SCLMAS command.

Example Program

Resulting Motion Profile:

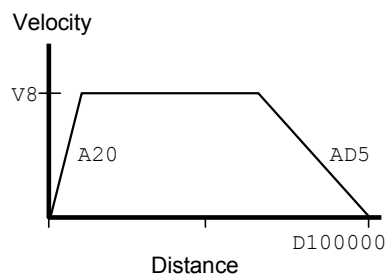


The program below controls two axes of motion. Axis 1 produces a preset (incremental) 100,000-count move in a nominally trapezoidal profile. Axis 2 produces a preset 80,000-count move in a nominally trapezoidal profile.

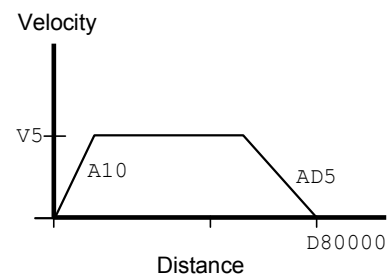
```

DEL BASIC          ; Delete program called BASIC
DEF BASIC          ; Begin definition of program called BASIC
MC00              ; Use the preset positioning mode for axes 1 & 2
MA00              ; Use the incremental (preset) positioning mode
A20,10            ; Accelerate axis 1 at 20 revs/sec/sec, and axis 2 at 10
AD5,5             ; Decelerate axis 1 and axis 2 at 5 revs/sec/sec
V8,5              ; Set axis 1 velocity to 8 revs/sec, and axis 2 to 5
D100000,80000    ; Set distance to 100,000 counts
GO11              ; Execute the motion on both axes
END                ; End definition of BASIC
  
```

Axis 1 profile:

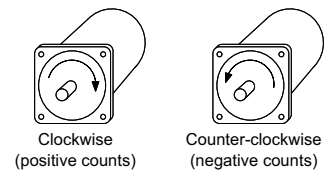


Axis 2 profile:



Direction of Motion
for Rotary Motors

Positive distance values (e.g., D20000) represent clockwise motion, negative values (e.g., D-20000) represent counter-clockwise motion. This assumes you connected the drive and motor (and feedback device for servo drives) according to the *Hardware Installation Guide* instructions.



Preset and Continuous Modes

The 6K controller can be programmed to position in either the preset (incremental or absolute) mode or the continuous mode. You should select the mode that will be most convenient for your application. For example, a repetitive cut-to-length application requires incremental positioning. X-Y positioning, on the other hand, is better served in the absolute mode. Continuous mode is useful for applications that require constant movement of the load based on internal conditions or inputs, not distance.

Positioning modes require acceleration, deceleration, velocity, and distance commands (*continuous mode does not require distance*). The table below identifies these commands and their units of measure, and which scaling command affects them.

Parameter	Units (Unscaled), Stepper	Units (Unscaled), Servo	Unit Scaling Command *
Acceleration	revs/sec ²	encoder/resolver: revs/sec ² ANI: volts/sec ²	SCLA (or SCLD **)
Deceleration	revs/sec ²	encoder/resolver: revs/sec ² ANI: volts/sec ²	SCLA (or SCLD **)
Velocity	revs/sec	encoder/resolver: revs/sec ANI: volts/sec	SCLV (or SCLD **)
Distance	steps	counts	SCLD ***

* Scaling must first be enabled with the `SCALE1` command. For details on scaling, see page 48.

** All Contouring and linear interpolated motion is scaled with `SCLD` (distance, accel/decel, and velocity).

*** An axis assigned as a master (for Following) is scaled by the `SCLMAS` command.

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 not allowed during on-the-fly changes
- Deceleration (AD) — s-curve deceleration not allowed during on-the-fly 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, see *On-The-Fly Motion Profiling* on page 151.

Preset Positioning Mode

A *preset* move is a point-to-point move of a specified distance. You can select preset moves by putting the 6K controller into preset mode (canceling continuous mode) using the MCØ command. Preset moves allow you to position the motor/load in relation to the previous stopped position (*incremental mode*—enabled with the MAØ command) or in relation to a defined zero reference position (*absolute mode*—enabled with the MA1 command).

Incremental Mode Moves

The incremental mode is the controller's default power-up mode. When using the Incremental Mode (MAØ), a preset move moves the motor/load the specified distance from its starting position. For example, if you start at position *N*, executing the D6ØØØ command in the MAØ mode will move the motor/load 6,000 units from the *N* position. Executing the D6ØØØ command again will move the motor/load an additional 6,000 units, ending the move 12,000 units from position *N*.

You can specify the direction of the move by using the optional sign + or - (e.g., D+6ØØØ or D-6ØØØ). Whenever you do not specify the direction (e.g., D6ØØØ), the unit defaults to the positive (+) direction.

```
Example SCALE0 ; Disable scaling
MA0 ; Set axis 1 to Incremental Position Mode
A2 ; Set axis 1 acceleration to 2 units/sec/sec
V5 ; Set axis 1 velocity to 5 units/sec
D4000 ; Set axis 1 distance to 4,000 positive units
GO1 ; Initiate motion on axis 1 (move 4,000 positive units)
GO1 ; Repeat the move
D-8000 ; Set distance to 8,000 negative units
; (return to original position)
GO1 ; Initiate motion on axis 1 (move 8,000 units in the negative
; direction and end at its original starting position)
```

Absolute Mode Moves

A preset move in the Absolute Mode (MA1) moves the motor/load the distance that you specify from the *absolute zero position*.

Establishing a Zero Position

One way to establish the zero position is to issue the PSET command when the load is at the location you would like to reference as absolute position zero (e.g., PSETØ, Ø defines the current position as absolute position zero for axes 1 and 2). Stepper axes with encoder feedback can use the PESET command set the absolute encoder position in ENCCNT1 mode.

The zero position is also established when the Go Home (HOM) command is issued, the absolute position register is automatically set to zero after reaching the home position, thus designating the home position as position zero.

The direction of an absolute preset move depends upon the motor's/load's position at the beginning of the move and the position you command it to move to. For example, if the motor/load is at absolute position +12,500, and you instruct it to move to position +5,000 (e.g., with the D5ØØØ command), it will move in the negative direction a distance of 7,500 steps to reach the absolute position of +5,000.

The 6K controller retains the absolute position, even while the unit is in the incremental mode. To ascertain the absolute position, use the TPC and PC commands.

```
Example SCALE0 ; Disable scaling
MA1 ; Set the controller to the absolute positioning mode
PSET0 ; Set axis 1 current absolute position to zero
A5 ; Set axis 1 acceleration to 5 units/sec/sec
V3 ; Set axis 1 velocity to 3 units/sec
D4000 ; Set axis 1 move to absolute position 4,000 units
GO1 ; Initiate axis 1 move (move to absolute position +4,000)
D8000 ; Set axis 1 move to absolute position +8,000
GO1 ; Initiate axis 1 move (starting from position +4,000, move 4,000
; additional units in the positive direction to position +8,000)
D0 ; Set axis 1 move to absolute position zero
GO1 ; Initiate axis 1 move (starting at absolute position +8,000,
; move 8,000 units in the negative direction to position zero)
```

Continuous Positioning Mode

The Continuous Mode (MC1) is useful in these situations:

- Applications that require constant movement of the load
- Synchronize the motor to external events such as trigger input signals
- Changing the motion profile after a specified distance or after a specified time period (T command) has elapsed

You can manipulate the motor movement with either buffered or immediate commands. After you issue the GO command, buffered commands are not executed unless the continuous command execution mode (COMEXC1 command) is enabled. Once COMEXC1 is enabled, buffered commands are executed in the order in which they were programmed. More information on the COMEXC mode is provided on page 15.

The command can be specified as *immediate* by placing an exclamation mark (!) in front of the command. When a command is specified as immediate, it is placed at the front of the command queue and is executed immediately.

```
Example A  COMEXC1      ; Enable continuous command processing mode
           COMEXS1     ; Allow command execution to continue after stop
           MC1         ; Sets axis 1 mode to continuous
           A10         ; Sets axis 1 acceleration to 10
           V1          ; Sets axis 1 velocity to 1
           GO1         ; Initiates axis 1 move (Go)
           WAIT(LEVEL=1) ; Wait to reach continuous velocity
           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 processing mode
           ; When the move is executed, the load will accelerate to 1 unit/sec,
           ; continue at that rate for 5 seconds, and then decelerate to a stop.

Example B  DEF progr1   ; Begin definition of program progr1
           COMEXC1     ; Enable continuous command processing mode
           COMEXS1     ; Allow command execution to continue after stop
           MC1         ; Set axis 1 to continuous positioning mode
           A10         ; Set axis 1 acceleration to 10
           V1          ; Set axis 1 velocity to 1
           GO1         ; Initiate axis 1 move (Go)
           WAIT(LEVEL=1) ; Wait for motor to reach continuous velocity
           T3          ; Time delay of 3 seconds
           A50         ; Set axis 1 acceleration to 50
           V10         ; Set axis 1 velocity to 10
           GO1         ; Initiate acceleration and velocity changes on axis 1
           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 processing mode
           END         ; End definition of program progr1
```

While in continuous mode, motion can be stopped if:

- You issue an immediate Stop (!S) or Kill (!K or ctrl/K) command.
- The load trips an end-of-travel limit switch or encounters a software end-of-travel limit.
- The load trips a registration input (a trigger input configured with the INFNCi-H command to function as a registration input).
- The load trips an input configured as a kill input (INFNCi-C or LIMFNCi-C) or a stop input (INFNCi-D or LIMFNCi-D).

NOTE

While the axis is moving, you cannot change the parameters of some commands (such as DRIVE and HOM). This rule applies during the COMEXC1 mode and even if you prefix the command with an immediate command identifier (!). For more information, see *Restricted Commands During Motion* on page 17.

End-of-Travel Limits

Related Commands

LHHard limit enable
LHADHard limit decel
LHADAHard limit decel (s)
LIMLVLLimit switch polarity
LSSoft limit enable
LSADSoft limit decel
LSADASoft limit decel (s)
LSNEGSoft limit (negative)
LSPOSSoft limit (positive)
TLIMHard limit status
TASFBits 15-18 indicate if
hard or soft limit
was encountered
TERFBit 2: hard limit hit
Bit 3: soft limit hit
(must enable ERROR
checking bits 2 & 3)
ERRORBit 2 or 3 is enabled,
the 6K will branch to
the ERRORP program
if a hard or soft limit
is encountered

The 6K controller can respond to both hardware and software end-of-travel limits. The purpose of hardware and software end-of-travel limits is to prevent the motor's load from traveling past defined limits. Software and hardware limits are typically positioned in such a way that when the software limit is reached, the motor/load will start to decelerate toward the hardware limit, thus allowing for a much smoother stop at the hardware limit. Software limits can be used regardless of incremental or absolute positioning. When a hardware or software end-of-travel limit is reached, the 6K controller stops that axis using the respective hardware deceleration rate (set with LHAD & LHADA) or software limit deceleration rate (set with LSAD & LSADA).

How to set up hardware end-of-travel limits (for each axis):

1. Connect the end-of-travel limit inputs according to the instructions in your 6K product's *Installation Guide*. To help assure safety, connect normally-closed switches and leave the active level at default "active low" setting (set with the LIMLVL command).
2. (Optional) Define the inputs to be used as end-of-travel inputs for the respective axes.
NOTE: When the 6K product is shipped from the factory, the inputs on the "LIMITS/HOME" connectors are factory-configured with the LIMFNC command to function as end-of-travel and home limits for their respective axes. If you intend to use digital inputs on an external I/O brick as limit inputs:
 - a. Assign the limit function to the external input with the INFNC command. For example, 1INFNC9-1R assigns the "axis 1 positive end-of-travel limit" function to the 1st input on SIM2 (I/O point 9) of I/O brick 1.
 - b. Reassign the respective "LIMITS/HOME" input to a non-limit function with the LIMFNC command. For example, LIMFNC1-A assigns the "general-purpose input" function to limit input 1 (normally assigned the "axis 1 positive end-of-travel limit" function).
3. Set the hard limit deceleration rate (LHAD & LHADA) to be used when the limit switch is activated. The LHADA command allows you to define an s-curve deceleration.
Stepper Axes: If your system is moving heavy loads or operating at high velocities, you can need to decrease the LHAD command value (deceleration rate) to prevent the motor from stalling (GEMINI and ZETA drives can compensate without reducing decel).

NOTES ON HARDWARE LIMITS

- 6K controllers are shipped from the factory with the hardware end-of-travel limits enabled, but not connected. Therefore, **motion will not be allowed until you do one of the following:**
 - Install limit switches or jumper the end-of-travel limit terminals to the GND terminal (see your product's *Installation Guide* for wiring instructions).
 - Disable the limits with the LH command (recommended only if the load is not coupled).
 - Reverse the active level of the limits by executing the LIMLVLO command for the respective axis.
- If you reverse the commanded direction polarity (CMDDIR1), you should swap the hardware end-of-travel switch connections to maintain a positive correlation with the commanded direction.

How to set up software end-of-travel limits (for each axis):

1. Use the LS command to enable the software end-of-travel limits for the appropriate axes (for example, LS1111 enables software limits for axes 1-4).
2. Define the positive-direction limit with the LSPOS command, and define the negative-direction limit with the LSNEG command. If you have scaling enabled (SCALE1), these limit values are scaled by the SCLD command. Both software limits can be defined with positive values (e.g., axis 2 in the example below)

NOTES ON SOFTWARE LIMITS

- The software limits (LSPOS & LSNEG) are referenced from a position of absolute zero. or negative values. *Care must be taken when performing incremental moves because the software limits are always defined in absolute terms.* They must be large enough to accommodate the moves, or a new zero reference position must be defined (using the PSET command) before each move.
- To ensure proper motion when using soft end-of-travel limits, be sure to set the LSPOS value to an absolute value greater than the LSNEG value.

Programming Example

In this sample of code (not a complete program), the hardware and software limits are enabled on axes 1 and 2, and disabled on axes 3 and 4. The distance scaling command (SCLD) is used to define software limit locations in revolutions from the absolute zero position (assumes a 4000 step/rev resolution). Deceleration rates are specified for both software and hardware limits. If a limit is encountered, the motors will decelerate to a stop.

```
; *****  
; These scaling setup commands are downloaded before the  
; limit setup commands are executed:  
SCALE1      ; Enable scaling  
@SCLD4000   ; Program soft limit distance in revs (all axes)  
@SCLA4000   ; Program soft limit accel/decel in revs/sec/sec (all axes)  
@ERES4000   ; Set encoder resolution to 4000 steps/rev (all axes)  
; *****  
LH3,3,0,0   ; Enable limits 1 and 2, disable limits 3 and 4  
LHAD10,10   ; Set hard limit deceleration  
LSAD5,10    ; Set soft limit deceleration  
LSNEG0,2    ; Set negative direction soft limit  
            ; (axis 1: 0 revs; axis2: 2 revs)  
LSPOS10,20  ; Establish positive soft limit  
            ; (axis 1: 10 revs; axis 1: 20 revs)  
LS3,3,0,0   ; Enable soft limits 1 and 2, disable limits 3 and 4
```

Homing (Using the Home Inputs)

Refer to the product's *Installation Guide* for instructions to wire hardware home limit switches.

Homing Status:

Status of homing moves is stored in bit #5 of the axis status register (indicates whether or not the home operation was successful). To display the status, use the `TASF` command or the `TAS` command. To use the status in a conditional expression (e.g., for an `IF` statement), use the `AS` assignment/comparison operator.

The *homing operation* is a sequence of moves that position an axis using the Home Limit input and/or the Z Channel input of an incremental encoder. The goal of the homing operation is to return the load to a repeatable initial starting location.

Zero Reference After Homing: As soon as the homing operation is successfully completed, the absolute position register is reset to zero, thus establishing a zero reference position (for servo axes using analog input feedback, this applies also to the voltage register).

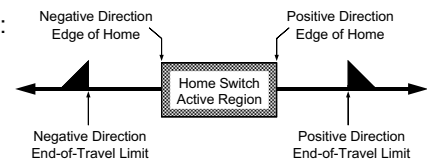
The homing operation has several potential homing functions you can customize to suit the needs of your application (illustrations of the effects of these commands are presented below):

Command	Homing Function (see respective command descriptions for further details)	Default
HOM.....	Initiate the homing move. To start the homing move in the positive direction, use <code>HOMØ</code> ; to home in the negative direction, use <code>HOM1</code> .	HOMx (do not home)
HOMA.....	Acceleration while homing.	HOMA1Ø (10 units/sec ²)
HOMAA.....	S-curve acceleration while homing.	HOMAA10 (10 units/sec ²)
HOMAD.....	Deceleration while homing.	HOMAD10 (10 units/sec ²)
HOMADA.....	S-curve deceleration while homing.	HOMADA10 (10 units/sec ²)
HOMBAC.....	Back up to home. The load will decelerate to a stop after encountering the active edge of the home region, and then will move in the opposite direction at the <code>HOMVF</code> velocity until the active edge of the home region is encountered. Allows the use of <code>HOMEDG</code> and <code>HOMDF</code> .	HOMBAC0 (function disabled)
HOMDF.....	Final approach direction—during backup to home (<code>HOMBAC</code>) or during homing to the Z channel input of an incremental encoder (<code>HOMZ</code>).	HOMDF0 (positive direction)
HOMEDG.....	Specify the side of the home switch on which to stop (either the positive-travel side or the negative-travel side).	HOMEGD0 (positive-travel side of switch)
LIMLVL.....	Define the home limit input active level (i.e., the state, high or low, which is to be considered an activation of the input). To use a normally-open switch, select active low (<code>LIMLVL0</code>); to use a normally-closed switch, select active high (<code>LIMLVL1</code>).	LIMLVL0 (active-low, use a normally-open switch)
HOMV.....	Velocity while seeking the home position (see also <code>HOMVF</code>).	HOMV1 (1 unit/sec)
HOMVF.....	Velocity while in final approach to home position—during backup to home (<code>HOMBAC</code>) or during homing to the Z channel input of an incremental encoder (<code>HOMZ</code>).	HOMVF.1 (0.1 unit/sec)
HOMZ.....	Home to the Z channel input from an incremental encoder. NOTE: The home limit input must be active prior to homing to the Z channel.	HOMZ0 (function disabled)

NOTES ABOUT HOMING

- Avoid using pause and resume functions during the homing operation. A pause command (`PS` or `!PS`) or pause/continue input (input configured with the `INFNCi-E` or `LIMFNCi-E` command) will pause the homing motion. However, when the subsequent resume command (`C` or `!C`) or pause/continue input occurs, motion will resume at the beginning of the homing motion sequence.

- Relevance of positive and negative direction:



- If an end-of-travel limit is encountered during the homing operation, the motion will be reversed and the home switch will be sought in the opposite direction. If a second limit is encountered, the homing operation will be terminated, stopping motion at the second limit.

Figures A and B show the homing operation when HOMBAC is not enabled. “CW” refers to the positive direction and “CCW” refers to the negative direction.

Figure A:

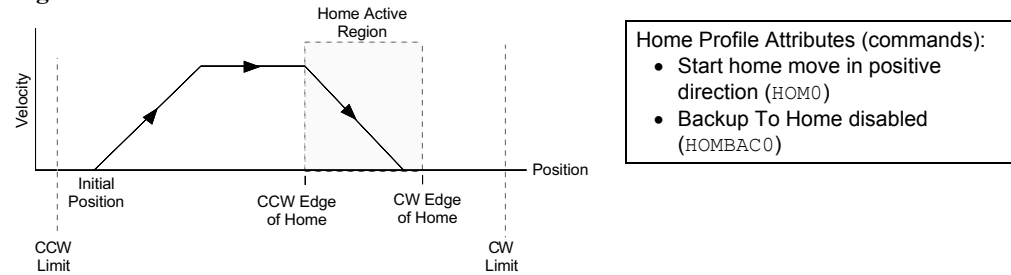
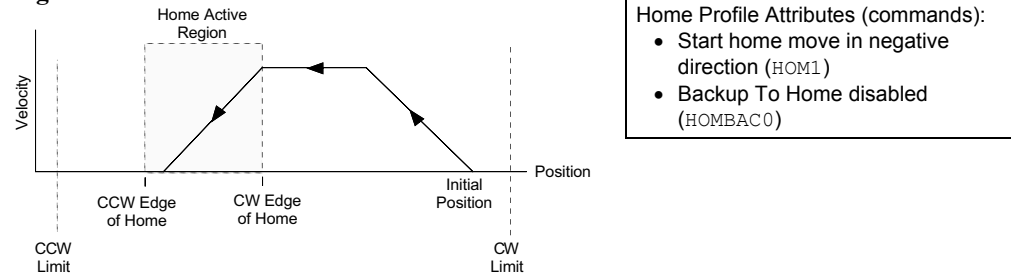


Figure B:



**Positive Homing,
Backup to Home
Enabled**

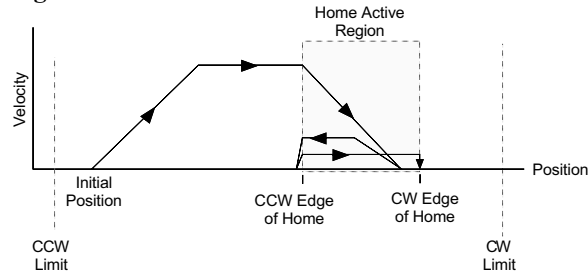
The seven steps below describe a sample homing operation when HOMBAC is enabled (see Figure C). The final approach direction (HOMDF) is CW and the home edge (HOMEDG) is the CW edge. “CW” refers to the positive direction and “CCW” refers to the negative direction.

NOTE

To better illustrate the direction changes in the backup-to-home operation, the illustrations in the remainder of this section show the backup-to-home movements with varied velocities. In reality, the backup-to-home movements are performed at the same velocity (HOMVF value).

- Step 1** A CW home move is started with the HOMØ command at the HOMA and HOMAA accelerations. Default HOMA is 10 revs (or volts or inches) per sec².
- Step 2** The HOMV velocity is reached (move continues at that velocity until home input goes active).
- Step 3** The CCW edge of the home input is detected, this means the home input is active. At this time the move is decelerated at the HOMAD and HOMADA command values. It does not matter if the home input becomes inactive during this deceleration.
- Step 4** After stopping, the direction is reversed and a second move with a peak velocity specified by the HOMVF value is started.
- Step 5** This move continues until the CCW edge of the home input is reached.
- Step 6** Upon reaching the CCW edge, the move is decelerated at the HOMAD and HOMADA command values, the direction is reversed, and another move is started in the CW direction at the HOMVF velocity.
- Step 7** As soon as the home input CW edge is reached, this last move is immediately terminated. The load is at home and the absolute position register is reset to zero.

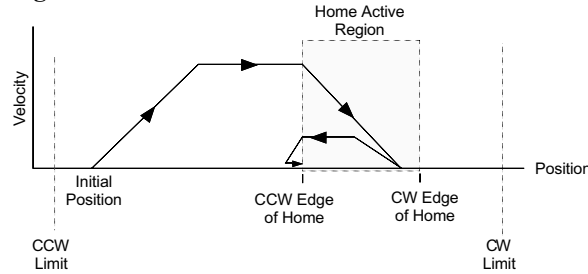
Figure C:



- Home Profile Attributes (commands):
- Start home move in positive direction (HOM0)
 - Backup To Home enabled (HOMBAC1)
 - Final approach direction is positive (HOMDF0)
 - Stop on the positive-travel side of the home switch active region (HOMEDG0)

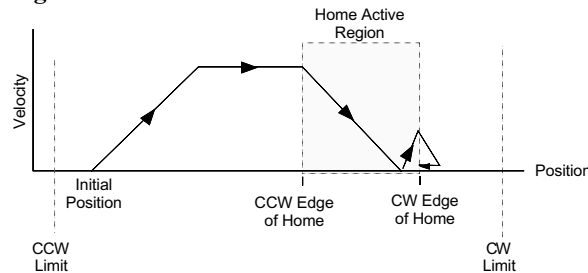
Figures D through F show the homing operation for different values of HOMDF and HOMEDG, when HOMBAC is enabled. “CW” refers to the positive direction and “CCW” refers to the negative direction.

Figure D:



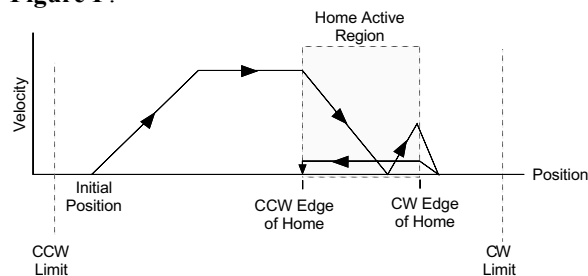
- Home Profile Attributes (commands):
- Start home move in positive direction (HOM0)
 - Backup To Home enabled (HOMBAC1)
 - Final approach direction is positive (HOMDF0)
 - Stop on the negative-travel side of the home switch active region (HOMEDG1)

Figure E:



- Home Profile Attributes (commands):
- Start home move in positive direction (HOM0)
 - Backup To Home enabled (HOMBAC1)
 - Final approach direction is negative (HOMDF1)
 - Stop on the positive-travel side of the home switch active region (HOMEDG0)

Figure F:

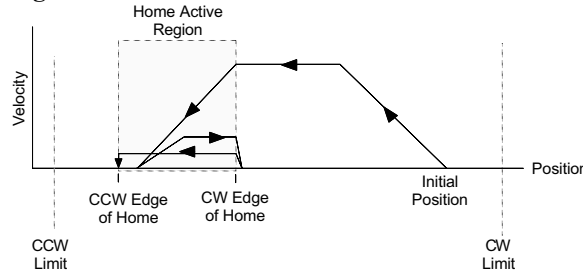


- Home Profile Attributes (commands):
- Start home move in positive direction (HOM0)
 - Backup To Home enabled (HOMBAC1)
 - Final approach direction is negative (HOMDF1)
 - Stop on the negative-travel side of the home switch active region (HOMEDG1)

Negative Homing, Backup to Home Enabled

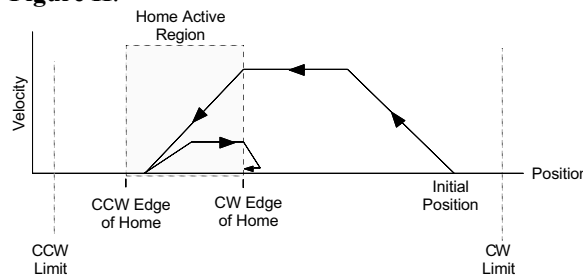
Figures G through J show the homing operation for different values of HOMDF and HOMEDG, when HOMBAC is enabled. "CW" refers to the positive direction and "CCW" refers to the negative direction.

Figure G:



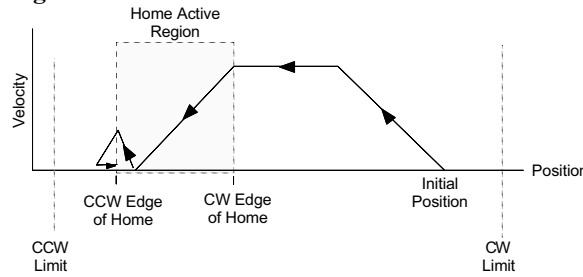
- Home Profile Attributes (commands):
- Start home move in negative direction (HOM1)
 - Backup To Home enabled (HOMBAC1)
 - Final approach direction is negative (HOMDF1)
 - Stop on the negative-travel side of the home switch active region (HOMEDG1)

Figure H:



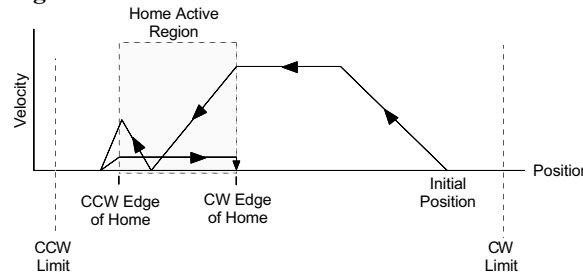
- Home Profile Attributes (commands):
- Start home move in negative direction (HOM1)
 - Backup To Home enabled (HOMBAC1)
 - Final approach direction is negative (HOMDF1)
 - Stop on the positive-travel side of the home switch active region (HOMEDG0)

Figure I:



- Home Profile Attributes (commands):
- Start home move in negative direction (HOM1)
 - Backup To Home enabled (HOMBAC1)
 - Final approach direction is positive (HOMDF0)
 - Stop on the negative-travel side of the home switch active region (HOMEDG1)

Figure J:

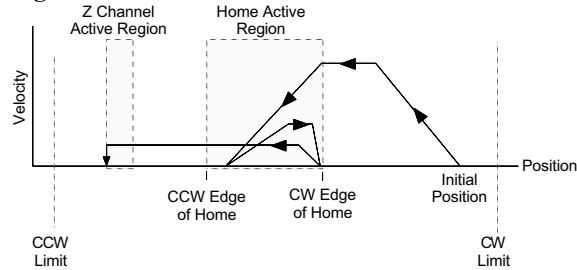


- Home Profile Attributes (commands):
- Start home move in negative direction (HOM1)
 - Backup To Home enabled (HOMBAC1)
 - Final approach direction is positive (HOMDF0)
 - Stop on the positive-travel side of the home switch active region (HOMEDG0)

Homing Using The Z-Channel

Figures K through O show the homing operation when homing to an encoder index pulse, or Z channel, is enabled (HOMZ1). The Z-channel will only be recognized after the home input is activated. It is desirable to position the Z channel within the home active region; this reduces the time required to search for the Z channel. “CW” refers to the positive direction and “CCW” refers to the negative direction.

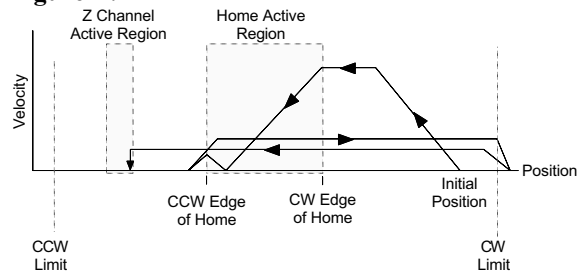
Figure K:



Home Profile Attributes (commands):

- Z-Channel homing enabled (HOMZ1)
- Start home move in negative direction (HOM1)
- Backup To Home enabled (HOMBAC1)
- Final approach direction is negative (HOMDF1)
- Stop on the negative-travel side of the z-channel active region (HOMEDG1)

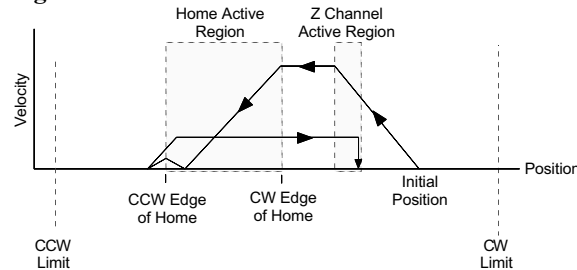
Figure L:



Home Profile Attributes (commands):

- Z-Channel homing enabled (HOMZ1)
- Start home move in negative direction (HOM1)
- Backup To Home enabled (HOMBAC1)
- Final approach direction is positive (HOMDF0)
- Stop on the positive-travel side of the z-channel active region (HOMEDG0)

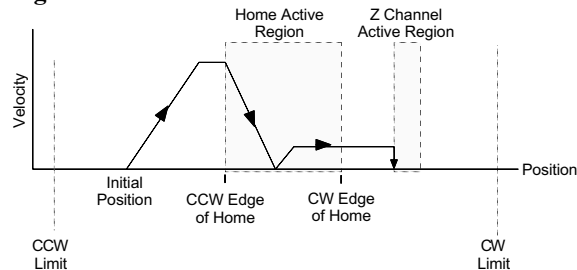
Figure M:



Home Profile Attributes (commands):

- Z-Channel homing enabled (HOMZ1)
- Start home move in negative direction (HOM1)
- Backup To Home enabled (HOMBAC1)
- Final approach direction is positive (HOMDF0)
- Stop on the positive-travel side of the z-channel active region (HOMEDG0)

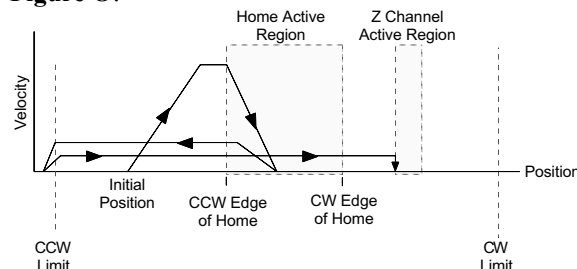
Figure N:



Home Profile Attributes (commands):

- Z-Channel homing enabled (HOMZ1)
- Start home move in positive direction (HOM0)
- Backup To Home disabled (HOMBAC0)
- Final approach direction is positive (HOMDF0)
- Stop on the positive-travel side of the z-channel active region (HOMEDG0)

Figure O:



Home Profile Attributes (commands):

- Z-Channel homing enabled (HOMZ1)
- Start home move in positive direction (HOM0)
- Backup To Home enabled (HOMBAC1)
- Final approach direction is positive (HOMDF0)
- Stop on the positive-travel side of the z-channel active region (HOMEDG0)

Encoder-Based Stepper Operation (stepper axes only)

When using an encoder in an stepper application, you can configure the following:

- Encoder resolution
- Stall Detection & Kill-on-Stall, Stall Deadband
- Encoder polarity
- Encoder-based position reference and position capture
- Encoder failure detection
- Commanded direction polarity

Encoder Resolution

You must specify the encoder resolution with the `ERES` command. The power-up default value for encoder resolution is 4,000 counts/rev. Listed below are the resolution values for Compumotor-supplied encoders.

- -RE, -RC, -EC, and -E Series Encoders:..... ERES4000
- -HJ Series Encoders:..... ERES2048
- Daedal Positioning Tables (encoder options):
 - E2 ERES42000
 - E3 ERES84000
 - E4 ERES420000
 - E5 ERES8400
- Dynaserv:

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

Stall Detection & Kill-on-Stall

To detect stalls without an encoder, see *Drive Stall Detection* on page 47.

The `ESTALL1` command enables the controller to detect encoder-based stall conditions.

NOTE: Encoder count reference must be enabled (`ENCCNT1`) before stall detect (`ESTALL`) can be used.

If used with *Kill-on-Stall* enabled (`ESK1` command), the move in progress will be aborted upon detecting a stall. If queried with the `ER` or the `AS` commands, the user can branch to any other section of program when a stall is detected. Refer to the `ER`, and `AS` command descriptions in the *6K Series Command Reference* for more information.

Kill-on-Stall functions only if the stall detection is enabled (`ESTALL1`).



WARNING



Disabling the Kill-on-Stall function with the `ESK0` command will allow the controller to finish the move regardless of a stall detection, even if the load is jammed. **This can potentially damage user equipment and injure personnel.**

Stall Deadband

Another encoder set-up parameter is the Stall Backlash Deadband (`ESDB`) command. This command sets the number of commanded counts of error allowed, after a change in direction, before a stall will be detected. This is useful for situations in which backlash in a system can cause false stall situations.

Encoder Set Up Example

The example below illustrates the features discussed in the previous paragraphs. The DRES statement defines the motor resolutions. The ERES statement defines the number of encoder steps per encoder revolution. Standard 1000-line encoders are used on all axes that produce 4000 quadrature steps/rev. Encoder counting (ENCCNT), stall detect (ESTALL) and kill-on-stall (ESK1) are enabled. If a stall is detected, the motor's movement is killed. The ESDB statement defines the stall deadband.

*Examples for 4 Axes
(command line samples)*

```
DRES25000,25000,25000,200 ; Set drive resolution
ERES4000,4000,4000,4000 ; Set encoder resolution
ESK11111 ; Enable kill motion on stall
ENCCNT1111 ; Enable encoder counting (this is
; required for ESTALL to work)
ESTALL1111 ; Enable stall detection
ESDB0,0,10,10 ; Set stall deadband
```

Encoder Polarity

If the encoder input is counting in the wrong direction, you can reverse the polarity with the ENCPOL command. 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).

NOTES

- 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.
- 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).

*Programming
Scenario (as seen in a
terminal emulator)*

```
; This programming scenario assumes the encoder polarity is reversed
; from the commanded direction (e.g., commanding a move of +10 units,
; yields and encoder position of -10 units).
;
> PSET0 ; Define current position of axis 1 as position zero
> 1TPE ; Check the position of encoder 1
*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 a distance of 8,000 units
> 1TPE ; Check the position of encoder 1
*1TPE-8000 (response shows that encoder 1 is at position -8000,
the minus sign indicates that the encoder is counting in the
wrong direction)

> DRIVE0 ; Disable the drive (disabled before changing polarity)
> ENCPOL1 ; Reverse encoder polarity on axis 1
> PSET0 ; Define current position of axis 1 as position zero
> DRIVE1 ; Enable the drive
> D+8000 ; Set distance to 8,000 units in the positive direction
> GO1 ; Move axis 1
> 1TPE ; Check the position of encoder 1
*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)
```

Encoder Count/Capture Referencing

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, axis 2 doesn't
INFNC1-H ; Configure trigger 1A as position capture input for axis 1
INFNC3-H ; Configure trigger 2A as 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.
```

Encoder Failure Detection

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 (default is disabled). For example, the `EFAIL1111000` command enables encoder failure checking on axes 1-4, but not axes 5-8.

The 6K detects a failure if both channel A+ and channel A- are the same state (i.e., both high or both low). The B channel is not checked. A+ and A- are internally pulled to +5V; therefore, if the encoder is disconnected a failure is detected. A failure could occur if either A+ or A- are disconnected, it depends on whether the other signal was high or low at that time. The typical causes for the failure would be a disconnected encoder or a failed encoder.

If `EFAIL` is enabled for an axis, and an encoder failure 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`). The error condition is cleared by reconnecting the encoder while `EFAIL` is enabled for that particular axis.

DO NOT USE WITH SINGLE-ENDED ENCODERS

Use the `EFAIL` feature only with differential encoders. Do not attempt to use the `EFAIL` feature with single-ended encoders; because the A- terminal is not connected, the 6K will always detect an encoder failure if `EFAIL` is enabled.

Commanded Direction Polarity

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 encoder. 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 encoder, or (b) change the sign of all the motion-related commands in your program.

NOTES

- The `CMDDIR` command cannot be executed while motion is in progress or while the drive is enabled. For example, you could wait for motion to be complete (indicated when `TAS` and `AS` bit 1 is a zero) and then use the `DRIVE` command to disable the appropriate axis before executing the `CMDDIR` command.
- Before changing the commanded direction polarity, make sure there is 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).
- 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` setting is automatically saved in non-volatile memory.

Servo Setup (servo axes only)

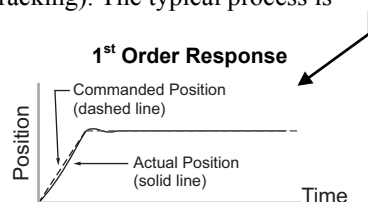
Use Motion Planner

Motion planner provides wizards and a tuning aide to create setup code for your servo product.

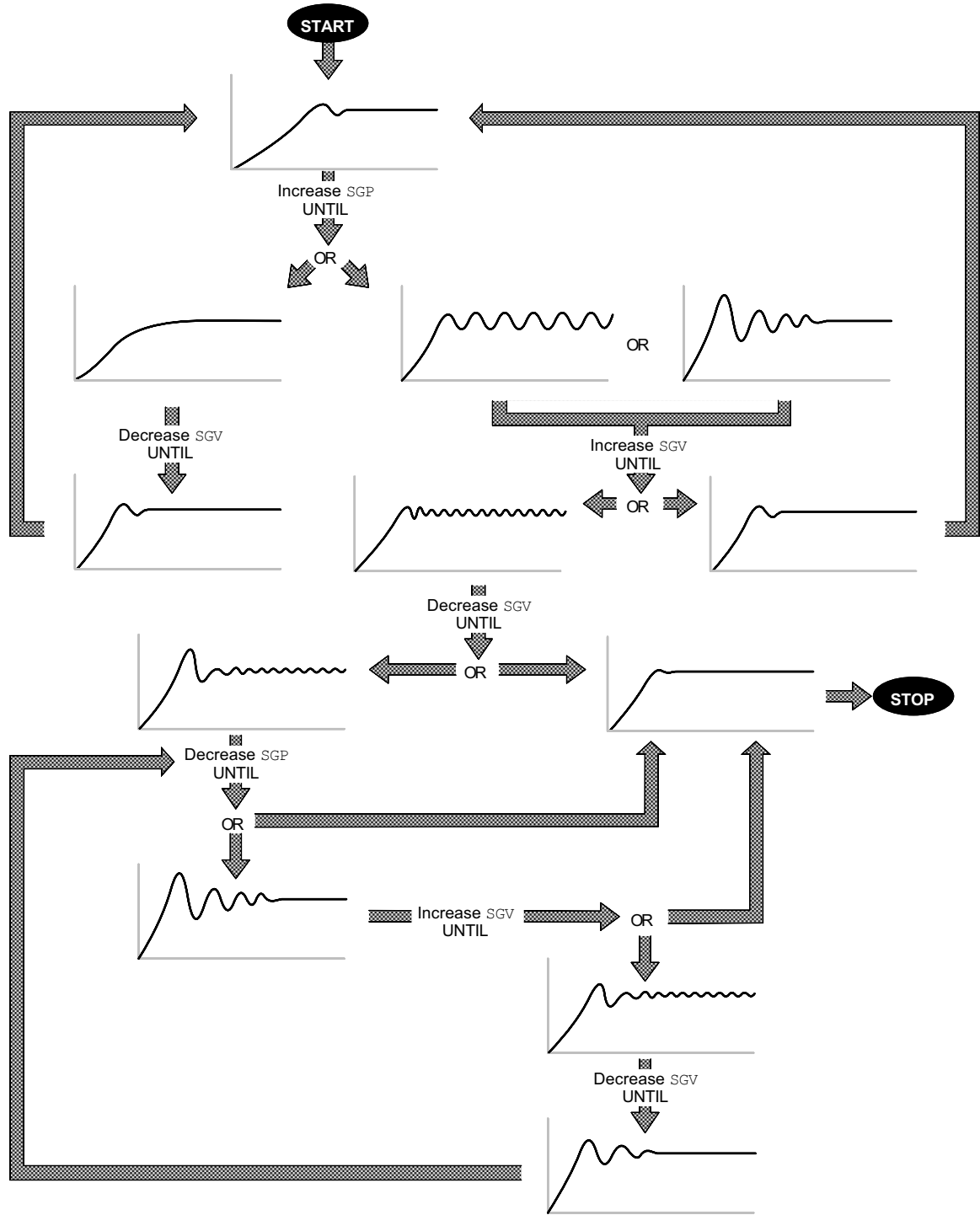
To assure optimum performance you should tune your servo system. The goal of the tuning process is to define the gain settings, servo performance, and feedback setup (see command list below) that you can incorporate into your application program. (Typically, these commands are placed into a setup program). Servo tuning should be performed as part of the application *setup process*, as described below. **To tune your servo system (4-step process):**

1. After starting Motion Planner, you will see the Editor window. Select the Servo Tuner tab. If a Servo Tuner session is not already open, on the File click New. Then select Servo Tuner and click Ok.
2. To send a pre-programmed step output to the drive, click Start. Notice that the graph display draws the commanded and actual velocity profiles so that you can graphically tune your servo system.

Optimize the proportional (SGP) and velocity (SGV) values by iteratively changing gains and viewing the results on the graph display. The object is to achieve a 1st order response (minimal overshoot and close position tracking). The typical process is illustrated in the flow diagram on the next page.



Basic Tuning Process



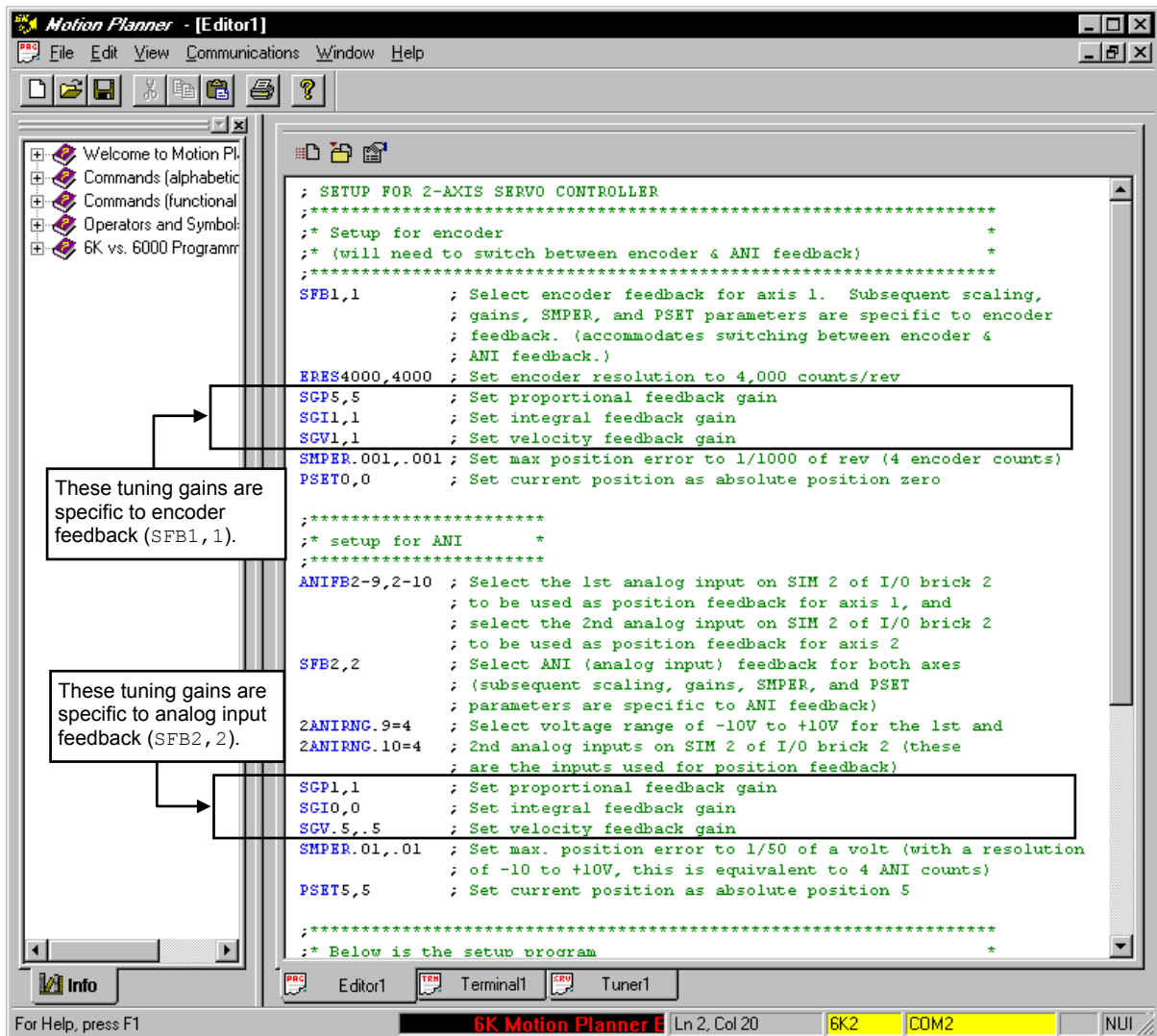
This figure outlines the basic process to achieve a 1st Order Response (minimal overshoot and close position tracking).

3. Repeat step 2 for each axis. Then write down the final gain settings.
4. When you have determined which tuning gains are best for your application's performance, insert the gain commands into your setup program:
 - a. Click the "Copy Gains" to Clipboard button. This copies the gain commands to your computer's clipboard.
 - b. Click the "Editor" tab to bring the program editor to the front.
 - c. Place the cursor at the location in your program where you wish to insert the gain commands (see note below).
 - d. Paste the gain commands at the location of the cursor.

NOTE

The tuning gains are specific to the feedback source selection in effect at the time the gain commands are executed. The factory default feedback source (selected with the SFB command) is encoder feedback. The illustration below demonstrates where to insert the gain commands relative to the SFB command.

If your application requires you to switch between feedback sources for the same axis, then for each feedback source you must select the source with the SFB command and then execute the tuning gain commands relevant to the feedback source (an example is provided in the illustration below).



Sample code generated by Motion Planner

Tuning-Related Commands (see 6K Series Command Reference for details)

Tuning Gains:	
SGP.....	Sets the proportional gain in the PIV&F servo algorithm.
SGI.....	Sets the integral gain in the PIV&F servo algorithm.
SGV.....	Sets the velocity gain in the PIV&F servo algorithm.
SGAF.....	Sets the acceleration feedforward gain in the PIV&F _a algorithm.
SGVF.....	Sets the velocity feedforward gain in the PIV&F _v algorithm.
SGILIM....	Sets a limit on the correctional control signal that results from the integral gain action trying to compensate for a position error that persists too long.
SGENB	Enables a previously-saved set of PIV&F gains. A set of gains (specific to the current feedback source selected with the SFB command) is saved using the SGSET command.
SGSET	Saves the presently-defined set of PIV&F gains as a <i>gain set</i> (specific to the current feedback source on each axis). Up to 5 gain sets can be saved and enabled at any point in a move profile, allowing different gains at different points in the profile.

Feedback Setup:	
SFB	Selects the servo feedback device (encoder or analog input). To use analog input feedback, you must first use the ANIFB command to configure the targeted analog input to be used for feedback. IMPORTANT: Parameters for scaling, tuning gains, max. position error (SMPER), and position offset (PSET) are specific to the feedback device selected (with the SFB command) at the time the parameters are entered (see programming examples in the 6K Programmer's Guide).
ERES	Encoder resolution.
SMPER.....	Sets the maximum allowable error between the commanded position and the actual position as measured by the feedback device. If the error exceeds this limit, the controller activates the Shutdown output and sets the DAC output to zero (plus any SOFFS offset). If there is no offset, the motor will freewheel to a stop. You can enable the ERROR command to continually check for this error condition (ERROR.12-1), and when it occurs to branch to a programmed response defined in the ERRORP program.

Encoder Polarity

To change the commanded and encoder polarity, use the CMDDIR command instead (see page 71 for details).

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 is counting in the wrong direction, you can reverse the polarity with the ENCPOL command. This allows you to reverse the counting direction without having to change the actual wiring to the encoder. 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 respective encoder will start counting in the opposite direction (including all the encoder position register, reported with TPE and PE). The polarity of the encoder is immediately changed whether or not the encoder is currently selected with the SFB command.

NOTES

- You cannot change the encoder on a specific axis while that axis is moving.
- Changing the encoder 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.
- 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 or page 71 for full details).

*Programming Scenario
(as seen when the
commands are typed
into a terminal
emulator)*

```
> 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         (Check the position of encoder 1)
*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         (Check the position of encoder 1)
*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         (Check the position of encoder 1)
*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)
```

Commanded Direction Polarity

EXAMPLE

The command to change the polarity for axis 2 is
`CMDDIR, 1`

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/valve and the feedback device, or (b) change the sign of all the motion-related commands in your program.

NOTES

- 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 `TAS` and `AS` bit 1 is a zero) and then use the `DRIVE` command to disable the appropriate axis before executing the `CMDDIR` command.
- 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).
- 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.

DAC Output Limits

If you will not be using the entire -10V to +10V range of the 6K controller's analog output, you can set up maximum (DACLIM) limits. For example, setting the DAC limit to 8.000V (DACLIM8.00) 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 to the drive.

Servo Control Signal Offset

The SOFFS command provides a means of setting the controller's analog output to a known voltage value. This could be useful for these occasions:

- Testing motion in an open-loop configuration (all gains set to zero).
- If the commanded output is set to zero (motor is supposed to be stationary), but it keeps moving, you can impose an offset value to stop motion. *This is the same effect as the balance input on most analog servo drives.*

Use the TDAC command to check the voltage being commanded at the servo controller's analog output (the voltage displayed includes and offset in effect).

WARNING — Torque Drive Users

If there is little or no load attached, the SOFFS offset can cause an acceleration to a high speed.

Servo Setup Example

This section shows examples of how the servo setup commands might be incorporated into a setup program. The example shows that much of the feedback selection and scaling code is placed in the program file before the setup program definition; this program file structure is required because scaling parameters are not allowed in a program.

USE THE SETUP WIZARD IN MOTION PLANNER

Motion Planner automatically generates the setup code when you use the wizards in the Editor. More information on creating and executing setup programs is provided on page 10.

```

; SETUP FOR 2-AXIS CONTROLLER
;*****
;* Setup for encoder *
;* (will need to switch between encoder & ANI feedback) *
;*****
SFB1,1 ; Select encoder feedback for axis 1. Subsequent scaling,
; gains, SMPER & PSET parameters are specific to encoder feed-
; back. (accommodates switching between encoder & ANI feedback)
ERES4000,4000 ; Set encoder resolution to 4,000 counts/rev
SCLA4000,4000 ; Allow programming accel/decel in revs/sec/sec
SCLV4000,4000 ; Set scaling for programming velocity in revs/sec
SCLD4000,4000 ; Set scaling for programming distances in revs
SGP5,5 ; Set proportional feedback gain
SGI1,1 ; Set integral feedback gain
SGV1,1 ; Set velocity feedback gain
SMPER.001,.001 ; Set max. position error to 1/1000 of rev (4 encoder counts)
PSET0,0 ; Set current position as absolute position zero

;*****
;* setup for ANI *
;*****
ANIFB2-9,2-10 ; Select the 1st analog input on SIM 2 of I/O brick 2
; to be used as position feedback for axis 1, and
; select the 2nd analog input on SIM 2 of I/O brick 2
; to be used as position feedback for axis 2
SFB2,2 ; Select ANI (analog input) feedback for both axes
; (subsequent scaling, gains, SMPER, and PSET
; parameters are specific to ANI feedback)
2ANIRNG.9=4 ; Select voltage range of -10V to +10V for the 1st and
2ANIRNG.10=4 ; 2nd analog inputs on SIM 2 of I/O brick 2 (these
; are the inputs used for position feedback)
SCLA205,205 ; Allow programming accel/decel in volts/sec/sec
SCLV205,205 ; Allow programming velocity in volts/sec
SCLD205,205 ; Allow programming distances in volts
SGP1,1 ; Set proportional feedback gain
SGI0,0 ; Set integral feedback gain
SGV.5,.5 ; Set velocity feedback gain
SMPER.01,.01 ; Set max. position error to 1/50 of a volt (with a resolution
; of -10 to +10V, this is equivalent to 4 ANI counts)
PSET5,5 ; Set current position as absolute position 5

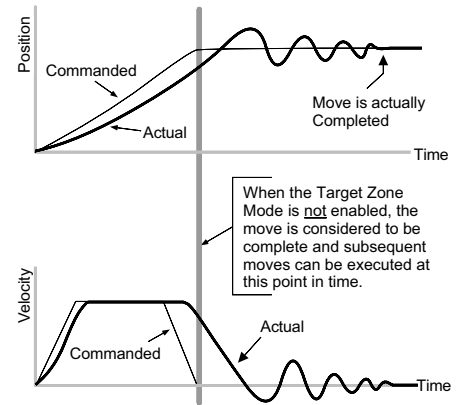
;*****
;* Below is the setup program *
;*****
DEF SETUP ; Begin definition of SETUP program
DRFEN11 ; Enable the drive fault input
DRFLVL11 ; Set drive fault level to active high
DRIVE00 ; Disable both drives
KDRIVE11 ; Invoke the Disable Drive On Kill feature, both axes
SFB1,1 ; Select encoder feedback for start of main program
Main ; Run the program called "main" (the main controlling
; program for this application)
END ; End definition of SETUP program

;*****
;* The following command (STARTP SETUP) is used to assign SETUP as the *
;* startup program to be automatically executed on power up or RESET. *
;*****
STARTP SETUP

```

Target Zone Mode (move completion criteria—servo axes only)

Under default operation (Target Zone Mode not enabled), the 6K product's move completion criteria is simply derived from the move trajectory. The 6K product considers the current preset move to be complete when the commanded trajectory has reached the desired target position; after that, subsequent commands/moves can be executed for that same axis. Consequently, the next move or external operation can begin before the actual position has settled to the commanded position (see diagram).



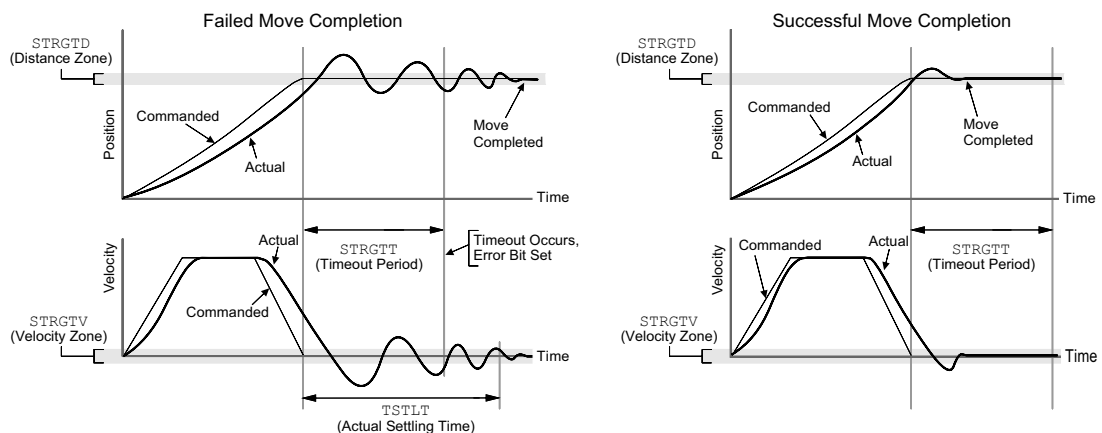
To prevent premature command execution before the actual position settles into the commanded position, use the *Target Zone Mode*. In this mode, enabled with the STRGTE command, the move cannot be considered complete until the actual position and actual velocity are within the *target zone* (that is, within the distance zone defined by STRGTD and less than or equal to the velocity defined by STRGTV). If the load does not settle into the target zone before the timeout period set with the STRGTT command, the 6K product detects a *timeout error* (see illustration below).

If the timeout error occurs, you can prevent subsequent command/move execution only if you enable the ERROR command to continually check for this error condition, and when it occurs to branch to a programmed response you can define in the ERRORP program. (Refer to the *Error Handling* section, page 30, for error program examples.)

As an example, setting the distance zone to ± 5 counts (STRGTD5), the velocity zone to ≤ 0.5 revs/sec (STRGTV0.5), and the timeout period to 1/2 second (STRGTT500), a move with a distance of 8,000 counts (D8000) must end up between position 7,995 and 8,005 and settle down to ≤ 0.5 rps within 500 ms (1/2 second) after the commanded profile is complete.

Damping is critical

To ensure that a move settles within the distance zone, it must be damped to the point that it will not move out of the zone in an oscillatory manner. This helps ensure the actual velocity falls within the target velocity zone set with the STRGTV command (see illustration below).



Checking the Settling Time

Checking the Actual Settling Time: Using the TSTLT command, you can 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. The TSTLT command is usable whether or not the Target Zone Settling Mode is enabled with the STRGTE command.

Programmable Inputs and Outputs (onboard and external inputs & outputs)

Programmable inputs and outputs allow the controller to detect and respond to the state of switches, thumbwheels, electronic sensors, and outputs of other equipment such as drives and PLCs. The I/O that can be used as programmable inputs and outputs are:

- Onboard I/O:
 - Limit inputs on the “LIMITS/HOME” connectors
 - Trigger inputs on the “TRIGGERS/OUTPUTS” connectors (pins 9, 11, 13, 15, 17, 19, 21 & 23). A “master trigger” is available (see “MASTER TRIG” terminal on the connector on top of the 6K chassis).
 - Digital outputs on the “TRIGGERS/OUTPUTS” connectors (1, 3, 5 & 7)
- Expansion I/O located on I/O bricks connected to the 6K controller’s “EXPANSION I/O” connector. Each I/O brick can hold from 1 to 4 of these I/O SIM modules in any combination (each SIM module provides 8 inputs or outputs, for a total of 32 I/O points per I/O brick):
 - Digital inputs
 - Digital outputs
 - Analog inputs
 - Analog outputs

USING THE STATE OF I/O TO CONTROL PROGRAMMED EVENTS: Based on the binary state of the inputs and outputs (binary status can be used in assignment/comparison operations using the LIM, IN and OUT operators), the controller can make program flow decisions and assign values to binary variables for subsequent mathematical operations. These operations and the associated program flow, branching, and variable commands are listed below.

Operation based on I/O State	Associated Commands	See Also*
I/O state assigned to a binary variable	LIM, [IN], [OUT], VARB	Variables (page 18)
I/O state used as a basis for comparison in conditional branching & looping statements	LIM, [IN], [OUT], IF, ELSE, NIF, REPEAT, UNTIL, WAIT, WHILE, NWHILE	Program Flow Control (page 23)
Input state used as a basis for a conditional GO	[IN], GOWHEN, LIM	Synchronizing Motion (page 159)
I/O state used as a basis for a program interrupt (GOSUB) conditional statement	ONIN	Program Interrupts (page 29)
Mimic PLC functionality by scanning I/O states with a compiled program	PLCP, SCANP	PLC Scan Mode (page 104)

* Refer also to the respective command descriptions in the *6K Series Command Reference*.

I/O UPDATE RATE: The programmable inputs and outputs are sampled at the “system update rate,” which is every 2 ms. **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). (If you disable the “Kill on I/O Disconnect” mode with KIOENØ, the 6K will not perform the kill.) 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.

Programmable I/O Bit Patterns

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.`

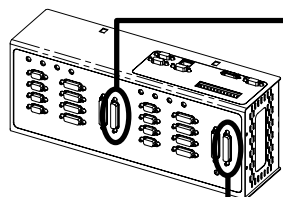
Bit 1 →
← Bit 17
- **Expansion I/O.** For example, the status command to check all digital inputs on I/O brick 2 is `2TIN`.
An example response for the 6K8 is: `*2TIN0010 0110_1100_0000_XXXX_XXXX_XXXX_XXXX.`

I/O Brick 2 →
← Bit 1
← Bit 32

Onboard Programmable 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:

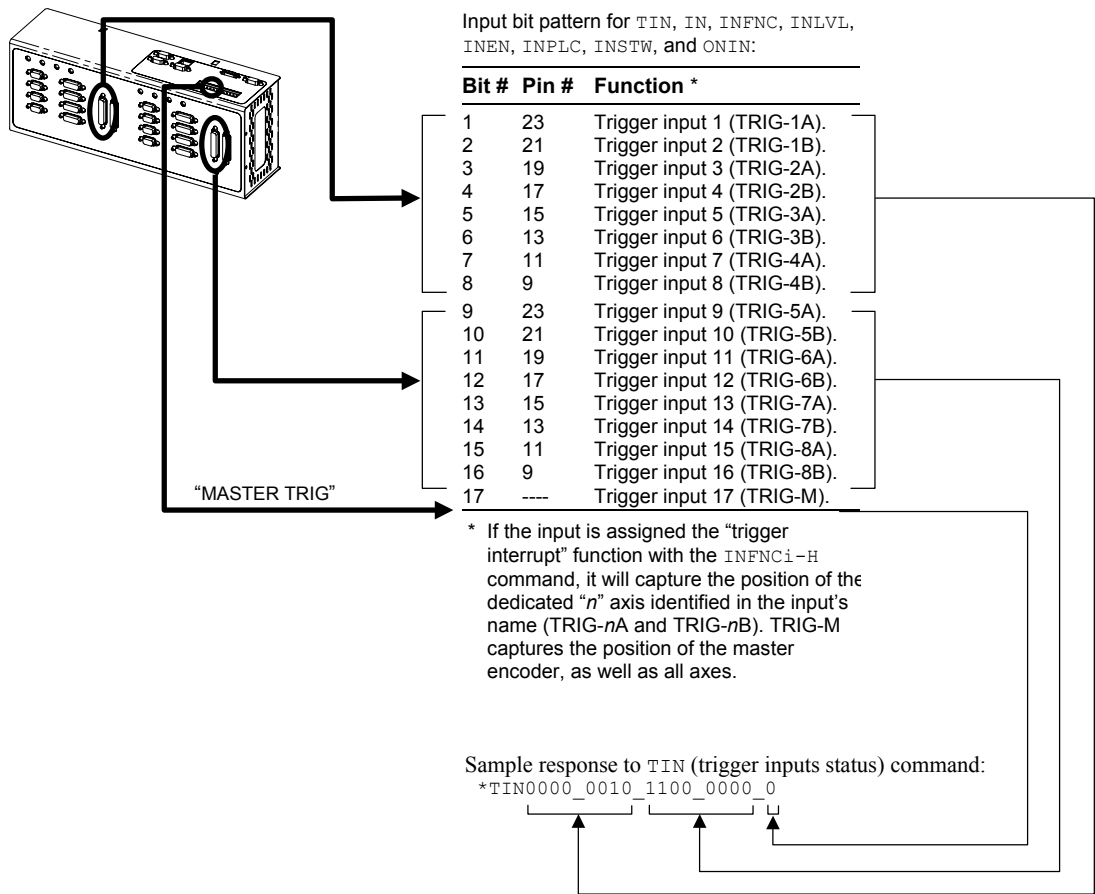
Bit #	Pin #	Function *
1	23	Positive end-of-travel limit, axis 1.
2	21	Negative end-of-travel limit, axis 1.
3	19	Home limit, axis 1.
4	17	Positive end-of-travel limit, axis 2.
5	15	Negative end-of-travel limit, axis 2.
6	13	Home limit, axis 2.
7	11	Positive end-of-travel limit, axis 3.
8	9	Negative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

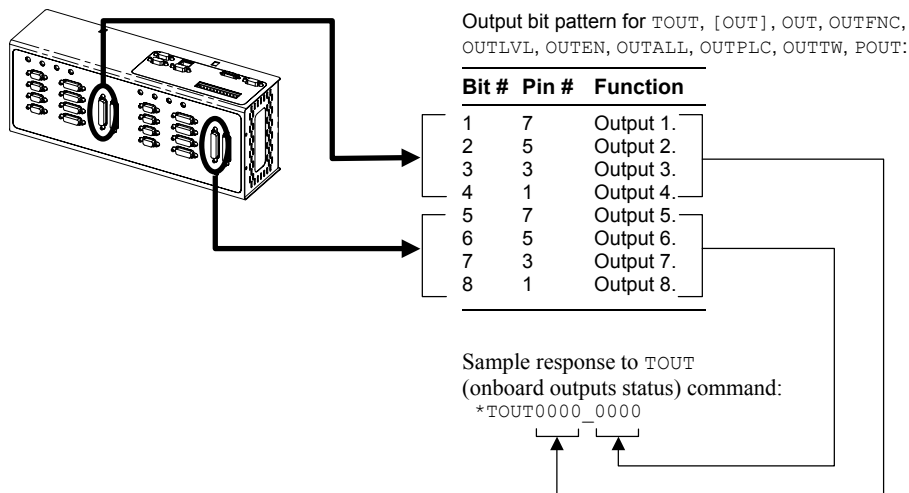
Sample response to TLIM (limit inputs status) command:

`*TLIM001_001_001_001_001_001_001_001`

Trigger Inputs ("TRIGGERS/OUTPUTS" connectors)



Outputs ("TRIGGERS/OUTPUTS" connectors)



Expansion I/O Bricks

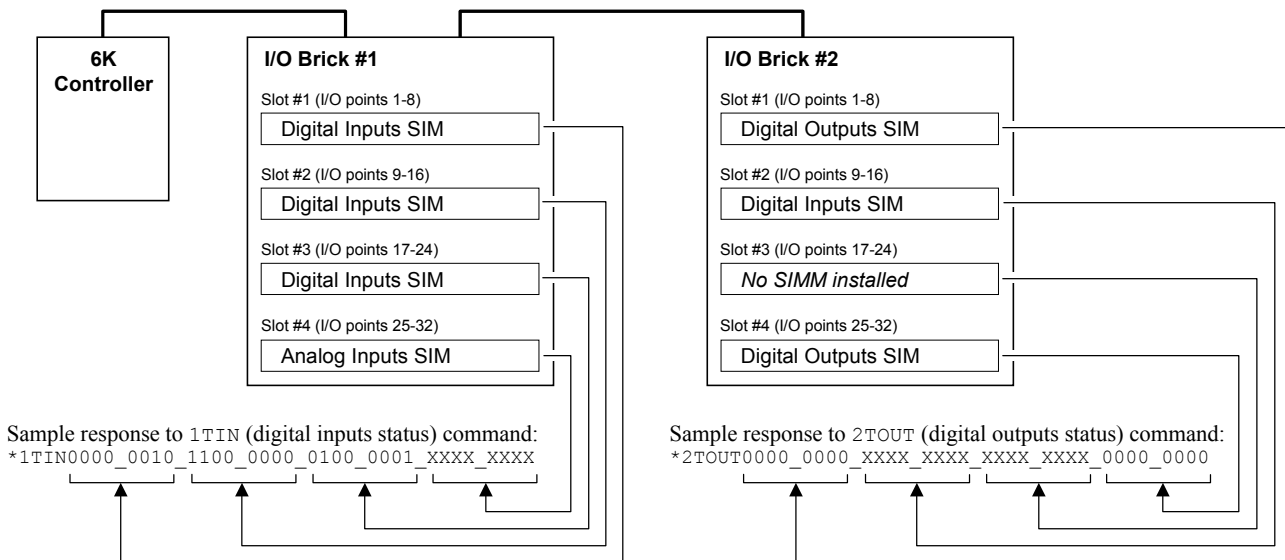
The 6K product allows you to expand your system I/O by connecting up to 8 I/O bricks (see *Installation Guide* for connections). Expansion I/O bricks can be ordered separately (referred to as the “EVM32”). Each I/O brick can hold from 1 to 4 of these I/O SIM modules in any combination:

SIM Type	Programming	Status Report, Assignment
Digital Inputs SIM (8 inputs)	INFNC, INLVL, INEN, ONIN, INPLC, INSTW	TIN, IN, TIO
Digital Outputs SIM (8 outputs)	OUT, OUTFNC, OUTLVL, OUTEN, OUTALL, OUTPLC, OUTTW, POUT	TOUT, [OUT], TIO
Analog Inputs SIM (8 inputs)	<ul style="list-style-type: none"> • Enable/Disable: ANIEN. • Voltage range: ANIRNG. • Joystick setup: JOYAXH, JOYAXL, JOYCDB, JOYCTR, JOYEDB, JOYZ. • Servo feedback: ANIFB, SFB • Following master source: ANIMAS, FOLMAS 	<ul style="list-style-type: none"> • Voltage: TANI, ANI, TIO • Servo position: TPANI, PANI, FB, TFB
Analog Outputs SIM (8 outputs)	ANO	TANO, [ANO], TIO

Each I/O brick has a unique “brick address”, denoted with the “” 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 assumes you are addressing the command to the onboard I/O.) Each I/O brick has 32 I/O addresses, referenced as absolute 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



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
```

Input Functions

Programmable input functions can be assigned to the onboard limit and trigger inputs, as well as to digital inputs on an expansion I/O brick connected to the 6K product. The appropriate input function command (LIMFNC or INFNC) depends on which input you are configuring:

- Onboard limit inputs (on the “LIMITS/HOME” connector) — use LIMFNC:

Syntax: LIMFNCi-<a>c

Number of the input. (see page 141 for input bit assignments) — points to *i*

Axis number (optional). Some functions may be made specific to one axis. When shipped from the factory, all limit inputs are assigned the limit function for the respective axis. For example, limit input 1 is assigned the positive travel end-of-travel limit function for axis 1 (LIMFNC1-1R). — points to *a*

Letter that selects the desired function (see list below). — points to *c*

- Onboard trigger inputs (on the “TRIGGERS/OUTPUTS” connector) and digital inputs installed on an expansion I/O brick — use INFNC:

Syntax: INFNCi-<a>c

I/O Brick number. Trigger inputs are considered collectively as I/O brick 0 (zero), and are addressed of the I/O brick number is left off the command. — points to *B*

Number of the input. (see page 143 for input bit assignments) — points to *i*

Axis number (optional). Some functions may be made specific to one axis. For example, to assign the 2nd input on SIMM2 (I/O point #9) of I/O brick 1 to be the stop input for axis 2 only, use the 1INFNC9-2D command. — points to *a*

Letter that selects the desired function (see list below). — points to *c*

Virtual Inputs

Virtual Inputs provide programming input functionality for data or external events that are not ordinarily represented by inputs. The Virtual Input Override (IN) command allows you to substitute almost any 32-bit data operand as a virtual input brick of 32 inputs (integer values are converted to binary). Page 8 provides a list of the data operands; the only data operands that are not allowed are: SIN, COS, TAN, ATAN, VCVT, SQRT, VAR, TW, READ, DREAD, DREADF, DAT, DPTR, and PI.

The virtual inputs behave similar to real inputs in that they are affected by INEN and INLVL, and they affect INFNC, INPLC, INSTW, INDUST, ONIN, and GOWHEN (IN=b<bbbb>) commands. Unlike real inputs, virtual inputs are not affected by the INDEB debounce setting.

NOTE: A virtual input can only be defined for expansion I/O bricks that are not connected on the serial I/O network (remember that up to 8 I/O bricks are allowed). For example, if your 6K unit has two I/O bricks, you can designate I/O bricks 3-8 as virtual I/O bricks.

EXAMPLE: Suppose a PLC is sending binary data via the VARB1 command to the 6K. If the binary state of VARB1 is assigned to input brick 2 (2IN=VARB1), the 6K can respond based on programmable input functions set up with the INFNC command.

```

2TIN          ; Brick 2 is not connected; therefore, the 6K will
              ; respond with an error message: "*INCORRECT I/O BRICK"
2IN=VARB1     ; Map the binary state of VARB1 to be the input state
              ; of "virtual" input brick 2 (2IN)
VARB1=b1010000 ; Change "virtual" input brick 2IN to a new VARB1 value
2TIN          ; Check the input status. The response will be:
              ; "*2IN1010_0000_0000_0000_0000_0000_0000"
    
```

Letter Designator	Function
A.....	General-purpose input (<u>default function for triggers & inputs on I/O bricks</u>)
B.....	BCD program select
C.....	Kill
<a>D.....	Stop (axis designator "a" is optional)
E.....	Pause/Continue
F.....	User fault
G.....	<RESERVED>
H.....	Trigger Interrupt for position capture or registration (trigger inputs only). Special trigger functions can be assigned with the TRGFN command (see page 162).
I.....	Cause an "Alarm Event" over the Ethernet interface
aJ.....	Jog in the positive-counting direction (axis designator is required)
aK.....	Jog in the negative-counting direction (axis designator is required)
aL.....	Jog velocity select (axis designator is required)
M.....	Joystick release
N.....	Joystick axis select
O.....	Joystick velocity select
P.....	One-to-one program select
Q.....	Program security
aR.....	End-of-travel limit for positive-counting direction (axis designator is required) *
aS.....	End-of-travel limit for negative-counting direction (axis designator is required) *
aT.....	Home limit (axis designator is required) *

* Limit inputs (on the "LIMITS/HOME" connector) are factory-set to their respective end-of-travel or home limit function (see page 57).

NOTES

- **Multi-tasking:** If the LIMFNC or INFNC command does not include the task identifier (%) prefix, the function affects the task that executes the LIMFNC or INFNC command. The only functions that can 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 can share the same input, but the input can only be assigned one function.
- **Limit of 32 functions per input group:** You can assign a maximum of 32 LIMFNC functions, and a maximum of 32 INFNC functions. For LIMFNC, this excludes functions A ("general-purpose") and R, S, and T (end-of-travel and home limit input functions). For INFNC, this excludes functions A ("general-purpose") and H ("trigger interrupt").

Input Status

* The purpose of the IN and LIM operators is to use the state of the inputs as a basis for conditional statements (IF, REPEAT, WHILE, GOWHEN, etc.) or for binary variable assignments (VARB). For examples, see the *Conditional Looping and Branching* section on page 25, and *Synchronizing Motion* on page 159.

Limit inputs (LIMFNC):

- Status display commands:
 - LIMFNC.....Active state and programmed function of all limit inputs
 - LIMFNCiSame as LIMFNC display, but only for the input number ("i")
 - TLIMHardware state of all limit inputs (binary report);
use TLIM.i to check the state of only one input ("i")
- Status assignment/comparison operator: *
 - LIM.....Hardware state (binary) of all limit inputs;
use LIM.i to check the state of only one input ("i")

Trigger inputs and digital inputs on expansion I/O bricks (INFNC):

- Status display commands:
 - TIO.....I/O brick configuration (which SIMs are present)
 - INFNC.....Active state and programmed function of all trigger inputs
 - INFNCi.....Same as INFNC display, but only for the trigger input number ("i")
 - INFNC.....Active state and programmed function of all inputs on I/O brick B
 - INFNCi.....Same as INFNC display, but only for the input number ("i") on brick B
 - TIN.....Hardware state of all onboard trigger inputs (binary report);
use TIN.i to check the state of only one input ("i")
 - TIN.....Hardware state of all digital inputs on I/O brick B (binary report);
use TIN.i to check the state of only one input ("i") on brick B
- Status assignment/comparison operator: *
 - INHardware state (binary) of triggers and expansion digital inputs;
use IN.i to check the state of only one input ("i") on brick B

Input Active Levels

Many people refer to a voltage level when referencing the state of programmable inputs and outputs. Using LIMLVL (for limit inputs) and INLVL (for triggers and I/O brick inputs), you can define the logic levels of the programmable inputs as positive or negative. The 6K product defaults to an input level of zero volts as its active level (referred to as “active low”); thus, a “1” will appear in a status command (TLIM & LIM, or TIN & IN) referencing an input state when the voltage level is zero volts.

Active Level Setting	State	LIM/TLIM or IN/TIN Report
Active Low, <u>the default setting</u> <ul style="list-style-type: none"> LIMLVL0 for limits INLVL0 for triggers/I/O bricks 	Grounded — sinking current (device drive the input is on)	1 (active)
	Not Grounded — not sinking current (device drive the input is off)	- 0 (inactive)
Active High <ul style="list-style-type: none"> LIMLVL1 for limits INLVL1 for triggers/I/O bricks 	Grounded — sinking current (device drive the input is on)	0 (inactive)
	Not Grounded — not sinking current (device drive the input is off)	- 1 (active)

Input Debounce Time

Using the INDEB command, you can change the input debounce time for programmable inputs. 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. For example, to set the debounce for all digital inputs on I/O brick 2 to 5 ms, use the 2INDEB6 command.

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 TRGLOT 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 I/O brick input or an onboard trigger input is assigned a limit input function (INFNCi-R, INFNCi-S, or INFNCi-T), the input will not be debounced.

“General Purpose”

- LIMFNCi-A
- INFNCi-A

This is the default function for INFNC inputs (triggers and I/O brick inputs). When an input is defined as a *General Purpose* input, the input is used as a standard input. You can then use this input to synchronize or trigger program events, through the use of the LIM or IN operator.

```

Example
DEL prog1      ; Precaution: Delete a program before defining it
DEF prog1      ; Begin definition of program prog1
INFNC1-A       ; No function (general purpose) for trigger input 1
INFNC2-A       ; No function (general purpose) for trigger input 2
INFNC3-D       ; Trigger input 3 is a stop input (all axes)
A10            ; Set acceleration
V10            ; Set velocity
D5             ; Set distance
WAIT(IN=b1XX)  ; Wait for onboard trigger input 1
GO1            ; Initiate motion
IF(IN=bX1)     ; If onboard trigger input 2
1TFB           ; Transfer feedback device position for axis 1
NIF            ; End IF statement
END            ; End definition of program prog1

```

BCD Program Select

- LIMFNCi-B
- INFNCi-B

The *BCD program select* function allows you to execute defined programs by activating the program select inputs.

BCD program select inputs are assigned BCD weights, with the least weight on the smallest numbered input. The next BCD weight is assigned to the next input defined as a BCD input. For example, the table to the right shows the BCD weights if inputs 1-8 are configured as program select inputs.

Input No.	BCD Weight
Input 1	1
Input 2	2
Input 3	4
Input 4	8
Input 5	10
Input 6	20
Input 7	40
Input 8	80

To execute a particular program, you activate the combination of inputs to achieve the BCD weight that corresponds to the *number of the program* (see note below). For example, if inputs 1-8 are defined as BCD inputs (as in the example above), activating inputs 4 and 6 would execute program 28, activating inputs 1 and 4 would execute program 9, and so on.

Program Numbers

A program's number is determined by the order in which the program was downloaded to the controller. The number of each program stored in the controller's memory can be obtained through the TDIR command — refer to the number reported in front of each program name. When selecting programs with BCD Program Select inputs, a program is executed when the total BCD weight of the active BCD inputs equals the program's number. When selecting programs with One-to-One Program Select inputs, the program number is assigned to one specific input and is executed when that input is activated.

Before you can execute programs using the BCD program select inputs, you must first enable scanning with the INSELP1 command. Once enabled, the controller will continuously scan the BCD inputs and execute the program (by number) according to the weight of the currently active BCD inputs. After executing and completing the selected program, the controller will scan the inputs again. **NOTE:** To disable scanning, enter !INSELP0 or place INSELP0 in a program that can be selected.

The INSELP command also determines how long the BCD program select input level must be maintained before the controller executes the program. This delay is referred to as debounce time (but is not affected by the INDEB setting).

Example

```
RESET          ; Return controller to power-up conditions
ERASE          ; Erase all programs
DEF PROG1      ; Begin definition of program PROG1
TPE           ; Transfer position of encoders
END           ; End program
DEF PROG2      ; Begin definition of program PROG2
TREV          ; Transfer software revision
END           ; End program
DEF PROG3      ; Begin definition of program PROG3
TSTAT         ; Transfer statistics
END           ; End program
INFNC1-B      ; Assign onboard input 1 as a BCD program select input
INFNC2-B      ; Assign onboard input 2 as a BCD program select input
INSELP1,50    ; Enable scanning inputs, levels must be maintained for 50ms
TDIR          ; Display number and name of programs stored in memory
; response from TDIR should be similar to following:
; *1 - PROG1 USES 6 BYTES
; *2 - PROG2 USES 18 BYTES
; *3 - PROG3 USES 99 BYTES
; *32877 OF 33000 BYTES (98%) PROGRAM MEMORY REMAINING
; *500 OF 500 SEGMENTS (100%) COMPILED MEMORY REMAINING
```



The number in front of each program name is the BCD weight required to execute the program.

You can now execute the programs by activating the correct combination of inputs:

- Activate trigger input 1 (BCD weight of 1) to execute program 1 (PROG1)
- Activate trigger input 2 (BCD weight of 2) to execute program 2 (PROG2)
- Activate trigger inputs 1 & 2 (BCD weight of 3) to execute program 3 (PROG3)

Kill

- LIMFNCi-C
- INFNCi-C

When an input defined as a *Kill* input goes active:

- Motion stops on all axes (using the LHAD and LHADA decel rate).
- Program currently in progress is terminated.
- Commands currently in the command buffer are eliminated.
- Drives are left in the enabled state (@DRIVE1), unless the “disable drive on kill” function is enabled with the KDRIVE command (see description on page 48).
- If error-checking bit 6 is enabled (e.g., ERROR . 6-1), the error status is reported by bit 6 of the TERF, TER and ER commands and the error program (assigned with the ERRORP command) will be executed to respond to the error condition.

NOTE: Kill is not intended as a means to temporarily inhibit motion; use the *Pause/Continue* input function instead (LIMFNCi-E or INFNCi-E).

Stop

- LIMFNCi-<a>D
- INFNCi-<a>D

An input defined as a *Stop* input will stop motion on one or all axes (see examples below). Deceleration is controlled by the programmed AD/ADA deceleration ramp. If error-checking bit 8 is enabled (e.g., ERROR . 8-1), the error status is reported by bit 8 of the TERF, TER and ER commands and the error program (assigned with the ERRORP command) will be executed to respond to the error condition.

After the Stop input is received, further program execution is dependent upon the COMEXS command setting:

COMEXS0: (default setting) Upon receiving a stop input, motion will stop, program execution will be terminated and cannot be resumed, and every command in the buffer will be discarded (exception: an axis-specific stop input does not dump the command buffer).

COMEXS1: Upon receiving a stop input, motion will stop, program execution will pause, and all commands following the command currently being executed will remain in the command buffer (*but the move in progress will not be saved*).

You can resume program execution (but not the move in progress) by issuing an immediate Continue (!C) command or by activating a pause/continue input (i.e., a general-purpose input configured as a pause/continue input with the INFNCi-E command—see below). *You cannot resume program execution while the move in progress is decelerating.*

COMEXS2: Upon receiving a stop input, the 6K responds as it does in the COMEXS0 mode, with the exception that you can still use the program-select inputs to select programs (INSELP value is retained). The program-select input functions are: BCD select (INFNCi-B or LIMFNCi-B; see page 82), and one-to-one select (INFNCi-P or LIMFNCi-P; see page 88). For further details on program selection with inputs, see INFNC (or LIMFNC) and INSELP.

Example 1 (stop all axes): The 3INFNC2-D command assigns the “stop” function to the 2nd input on SIM1 of I/O brick 3. When this input is activated, all axes will stop.

Example 2 (stop a specific axis): The 3INFNC2-4D command assigns the “stop” function to the 2nd input on SIM1 of I/O brick 3, but makes it specific only to the motion on axis 4. When this input is activated, only the motion on axis 4 axes will stop.

Pause/Continue

- LIMFNCi-E
- INFNCi-E

An input defined as a *Pause/Continue* input will affect motion and program execution depending on the COMEXR command setting, as described below. In both cases, when the input is activated, the current command being processed will be allowed to finish executing before the program is paused.

COMEXR0: Upon receiving a pause input, only program execution will be 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. Releasing the pause input or issuing a !C command will resume motion and program execution. *You cannot resume program execution while the move in progress is decelerating.*

User Fault

- LIMFNCi-F
- INFNCi-F

An input defined as a *User Fault* input acts as an immediate Kill (!K) command, stopping motion on all axes and terminating program execution. Motion is stopped at the rate set with the hard limit (LHAD & LHADA) commands.

If error-checking bit 7 is enabled (e.g., ERROR. 7-1), then a user fault input will cause a branch to the ERRORP error program (for more information, see *Error Handling* on page 30) and the occurrence of a user fault input will be reported by error bit 7 (see TERF, TER and ER commands).

Trigger Interrupt (INFNCi-H)

This function is available only for onboard trigger inputs. Any trigger input can be defined as a Trigger Interrupt input and can be used for these functions:

- Position Capture (see below)
- Special trigger functions assigned with the TRGFN command (see below)
- Registration (see discussion on page 155)

Notes About Trigger Interrupt Inputs

- The trigger interrupt input is debounced for 24 ms (default) before another input on the same trigger is recognized. If your application requires a different debounce time, you can change it with the TRGLOT command (see *Input Debounce Time* on page 81).
- When configured as Trigger Interrupts, the triggers cannot be affected by the input enable (INEN) command.
- 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). Each TTRIG/TRIG bit is cleared when the respective 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.

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 TRIG-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”, can 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	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.

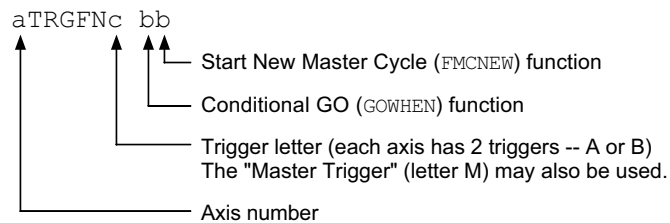
** If scaling is enabled, the captured position is scaled by SCLD or SCLMAS.

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 μ s between activating the trigger interrupt input and capturing the position; therefore, the accuracy of the captured position is equal to 50 μ s 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 can 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.

Trigger Functions

The Trigger Functions command (TRGFN) allows you to assign additional functions to trigger inputs that have been defined as trigger interrupt inputs with the INFNCi-H command. These trigger functions are cleared once the function is triggered. Command syntax is:



- “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)
 - 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 or a kill command 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; see GOWHEN command or page 159 for details.

- “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. Refer to the FMCNEW command or to page 184 for more on master cycles.

Code Examples

```
INFNC2-H      ; Assign trigger 1B (onboard input 2) to
              ; function as a trigger interrupt input.
1TRGFNBx1    ; When trigger 1B goes active, axis 1 will begin
              ; a new master cycle
2TRGFNB1     ; 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 becomes active.

Registration

If registration is enabled (with the RE command), activating a trigger interrupt input will initiate registration move(s) defined with the REG command. Refer to page 155 for details on the Registration feature.

Alarm Event

- LIMFNCi-I
- INFNCi-I

An input specified as an *Alarm Even* input 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 Communications Server features, see the *Com6srvr User's Guide for Gemini & 6K Series Products*.

Jogging the Motor

- LIMFNCi-aJ
- LIMFNCi-aK
- LIMFNCi-aL
- INFNCi-aJ
- INFNCi-aK
- INFNCi-aL

In some applications, you might want to manually move (*jog*) the load. You can configure these jog-related input functions:

“J”Jog in the positive counting direction when the input is active, stop when the input is inactive.

“K”Jog in the negative counting direction when the input is active, stop when the input is inactive.

“L”Select the high (JOGVH) or low (JOGVL) velocity setting for jog motion. Activating the input selects high velocity, deactivating the input selects low velocity.

The jog profile is defined with these commands listed below. **NOTE:** If scaling is enabled (SCALE1) the velocity is scaled by SCLV and accel/decel is scaled by SCLA.

JOGVHHigh velocity range for jogging. The high velocity is used when the jogging speed-select input (configured with INFNCi-aL) is active.

JOGVLLow velocity range for jogging. The low velocity is used when the jogging speed-select input (configured with INFNCi-aL) is inactive.

JOGA.....Jog acceleration

JOGAAJog acceleration (s-curve profile)

JOGADJog deceleration

JOGADA.....Jog deceleration (s-curve profile)

Once you set up the jog functions and move profile, you can attach a switch to the designated jog inputs and perform jogging. (Jog motion will not occur unless Jog Mode is enabled with the JOG command.) The example below shows you how to define a program to set up jogging.

Example

Step 1 Define program for jog setup:

```
DEF prog1      ; Begin definition of program prog1
JOGA25        ; Jog acceleration to 25 units/sec/sec
JOGAD25       ; Jog deceleration to 25 units/sec/sec
JOGVL.5       ; Low-speed jog velocity to 0.5 units/sec
JOGVH5        ; High-speed jog velocity to 5 units/sec
INFNC1-1J     ; Trigger input 1 is a positive-direction jog input, axis 1
INFNC2-1K     ; Trigger input 2 is a negative-direction jog input, axis 1
INFNC3-1L     ; Trigger input 3 is a speed-select input, axis 1
JOG1          ; Enable Jog function for axis 1
END           ; End program definition
```

Step 2 Download and run the prog1 program.

Step 3 Activate trigger input 1 to move the load on axis 1 in the positive direction at a velocity of 0.5 units/sec (until trigger input 1 is released). Deactivate the input to stop the axis.

- Step 4** Activate trigger input 2 to move the load on axis 1 in the negative direction at a velocity of 0.5 units/sec (until trigger input 2 is released). Deactivate the input to stop the axis.
- Step 5** Activate trigger input 3 to switch to high-speed jogging.
- Step 6** Repeat steps 3 and 4 to perform high-speed jogging at the JOGVH value (5 rps).

Joystick Functions

- LIMFNCi-M
- LIMFNCi-N
- LIMFNCi-O
- INFNCi-M
- INFNCi-N
- INFNCi-O

As part of the joystick setup process, you can configure programmable inputs to serve the joystick input functions listed below. Full details on joystick setup and operation are provided on page 114.

“M”The *Joystick Release* input signals the controller to end joystick operation and resume program execution with the next statement in your program. When the input is open (high, sinking current), 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, not sinking current), joystick mode can be enabled with the JOY command. The general 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”The *Joystick Axis Select* input 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, sinking current) selects the JOYAXH configuration. Closing the Axis Select input (input is low, not sinking current) selects the JOYAXL configuration.

“O”The *Joystick Velocity Select* input allows you to select the velocity for joystick motion. The JOYVH and JOYVL commands establish the high-speed velocity and the low-speed velocity, respectively. Opening the Velocity Select input (input is high, sinking current) selects the JOYVH configuration. Closing the Velocity Select input (input is low, not sinking current) selects the JOYVL configuration. The high range could be used to quickly move to a location, the low range could be used for accurate positioning. **NOTE:** When this input is not connected, joystick motion always uses the JOYVL velocity setting.

One-to-One Program Select

- LIMFNCi-iP
- INFNCi-iP

An input defined as a *One-to-One Program Select* input is assigned to execute one specific program. The targeted program is reference by its *number* (see note).

Program Numbers

A program's number is determined by the order in which the program was downloaded to the controller. The number of each program stored in the controller's memory can be obtained through the TDIR command — refer to the number reported in front of each program name. When selecting programs with One-to-One Program Select inputs, the program number is assigned to one specific input and is executed when the input is activated (see example below).

Before you can execute programs using the One-to-One program select inputs, you must first enable scanning with the INSELP2 command. Once enabled, the controller will continuously scan the inputs and execute the program (by number) according to the program number assigned to the input. After executing and completing the selected program, the controller will scan the inputs again. **NOTE:** To disable scanning, enter !INSELPØ or place INSELPØ in a program that can be selected.

The INSELP command also determines how long the program select input level must be maintained before the controller executes the program. This delay is referred to as debounce time (but is not affected by the INDEB setting).

```
Example  RESET          ; Return controller to power-up default conditions
        DEF proga      ; Begin definition of program proga
        TFB           ; Transfer position of feedback devices
        END           ; End program
        DEF progb     ; Begin definition of program progB
        TREV          ; Transfer software revision
        END           ; End program
        DEF progC     ; Begin definition of program progC
        TSTAT         ; Transfer statistics
        END           ; End program
        TDIR          ; Response should show:  *1 - PROGA USES 36 BYTES
        ;                                           *2 - PROGB USES 70 BYTES
        ;                                           *3 - PROGC USES 133 BYTES
        INFNC4-1P     ; Trigger input 4 (TRIG-2B) will select proga
        INFNC5-2P     ; Trigger input 5 (TRIG-3A) will select progB
        INFNC6-3P     ; Trigger input 6 (TRIG-3B) will select progC
        INSELP2,50    ; Enable scanning of inputs with a strobe time of 50 ms
```

You can now execute programs by making a contact closure from an input to ground to activate the input:

- Activate trigger input 4 to execute program 1 (proga)
- Activate trigger input 5 to execute program 2 (progb)
- Activate trigger input 6 to execute program 3 (progC)

Program Security

- LIMFNCi-Q
- INFNCi-Q

Once an input is assigned the *Program Security* function, the Program Security feature is enabled. The program security feature denies you access to the DEF, DEL, ERASE, MEMORY, LIMFNC and INFNC commands until you activate the Program Security 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 security input is not activated), you will receive the error message *ACCESS DENIED.

For example, once you issue the 2INFNC7-Q command, input 7 on I/O brick 2 (2IN.7) is assigned the program security function and access to the DEF, DEL, ERASE, MEMORY, LIMFNC and INFNC commands will be denied until you activate 2IN.7.

To regain access to the DEF, DEL, ERASE, MEMORY, LIMFNC or INFNC commands without the use of the program security 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.

NOTE: If you wish the Program Security feature to be enabled on power-up, place the INFNCi-Q or LIMFNCi-Q command in the start-up program (STARTP).

Limit Functions

- LIMFNCi-aR
- LIMFNCi-aS
- LIMFNCi-aT
- INFNCi-aR
- INFNCi-aS
- INFNCi-aT

The inputs on the “LIMITS/HOME” connectors are factory-configured with the LIMFNC command to function as end-of-travel and home limits for the respective axes (see illustration on page 76). The limit functions are:

“R” *Positive-Direction End-of-Travel Limit* input, axis specific.

“S” *Negative-Direction End-of-Travel Limit* input, axis specific.

“T” *Home Limit* input, axis specific.

If you intend to use digital inputs on an external I/O brick as limit inputs:

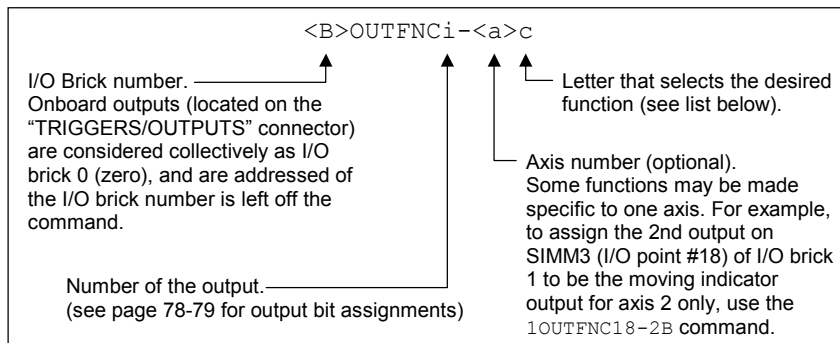
1. Assign the limit function to the external input with the INFNC command. For example, 1INFNC9-1R assigns the “axis 1 positive end-of-travel limit” function to the 1st input on SIM2 (I/O point 9) of I/O brick 1.
2. Reassign the respective “LIMITS” input to a non-limit function with the LIMFNC command. For example, LIMFNC1-A assigns the “general-purpose input” function to limit input 1 (normally assigned the “axis 1 positive end-of-travel limit” function).

NOTE: Once a trigger or I/O brick 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.

Output Functions

The 6K product provides programmable digital outputs, found on the “TRIGGERS/OUTPUTS” connect. Additional digital outputs can be installed on expansion I/O bricks (see example on page 78).

You can turn the controller's programmable outputs on and off with the Output (OUT, POUT or OUTALL) commands, or you can use the Output Function (OUTFNC) command to configure them to activate based on seven different situations. The OUTFNC syntax is as follows:



Letter Designator	Function
A.....	General-purpose output (default function)
<a>B.....	Moving/not moving (axis designator optional)
C.....	Program in progress
<a>D.....	Hardware or software end-of-travel limit encountered (axis designator optional)
<a>E.....	Stall indicator (axis designator optional) — stepper axes only
F.....	Fault indicator (indicates drive fault input or user fault input is active)
G.....	Position error exceeds maximum limit set with <code>SMPER</code> — servo axes only
H.....	Output on position

NOTES

- **Multi-tasking:** If the `OUTFNC` command does not include the task identifier (%) prefix, the function affects the task that executes the `OUTFNC` command. “Program in progress” (function C) is only function that can be directed to a specific task with %. Multiple tasks can share the same output, but the output can only be assigned one function.
- **Limit of 32 output functions:** You can assign a maximum of 32 `OUTFNC` functions. This excludes function A (“general-purpose”).

Output Status

Below is a list of the status commands you can use to ascertain the current state and/or defined function of the outputs.

- Status display commands:
 - `TIO`.....I/O brick configuration (which SIMs are present, including sinking/sourcing)
 - `OUTFNC`.....Active state and programmed function of all onboard outputs
 - `OUTFNCi`Same as `OUTFNC` display, but only for the output number (“i”)
 - `OUTFNC`.....Active state and programmed function of all outputs on I/O brick B
 - `OUTFNCi`Same as `OUTFNC` display, but only for the output number (“i”) on I/O brick B
 - `TOUT`Hardware state of all onboard outputs (binary report);
use `TOUT.i` to check the state of only one output (“i”)
 - `TOUT`Hardware state of all outputs (binary report) on I/O brick B;
use `TOUT.i` to check the state of only one output (“i”) on brick B

- Status assignment/comparison operator: *
 - OUTHardware state (binary) of all onboard outputs;
use OUT . i to check the state of only one output (“i”)
 - OUTHardware state (binary) of all outputs on I/O brick B;
use OUT . i to check the state of only one output (“i”) on I/O brick B
- * The purpose of the OUT operator is to use the state of the outputs as a basis for conditional statements (IF, REPEAT, WHILE, GOWHEN, etc.) or for binary variable assignments (VARB).

Output Active Levels

Using OUTLVL, you can define the logic levels of the programmable outputs as positive or negative. OUTLVL0 selects active low (default setting); OUTLVL1 selects active high.

Onboard Outputs (on the “TRIGGERS/OUTPUTS” connector):

OUTLVL Setting	OUT State *	Factory Default (use internal 24V)	Use an external supply at VINref	OUT/TOUT status
OUTLVL0 (default)	OUT1	Sinking current	Sinking current	1
OUTLVL0 (default)	OUT0	Sourcing 24V @ 0.7mA	Sourcing VINref	0
OUTLVL1	OUT1	Sourcing 24V @ 0.7mA	Sourcing VINref	1
OUTLVL1	OUT0	Sinking current	Sinking current	0

* The output is “active” when it is commanded by the OUT, OUTP, or POUT command (for example, OUTxx1 activates output 3).

Outputs on Expansion I/O Bricks:

- Sinking vs. Sourcing Outputs. On power up, the 6K controller auto-detects the type of output SIM installed on each external I/O brick, and automatically changes the OUTLVL setting accordingly. If sinking (NPN) outputs are detected, OUTLVL is set to active low (OUTLVL0); if sourcing (PNP) outputs are detected, OUTLVL is set to active high (OUTLVL1).
- Disconnect I/O Brick. 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. (If you disable the “Kill on I/O Disconnect” mode with KIOENØ, the 6K will not perform the kill.) 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.
- Relationships:

Output Type	OUTLVL Setting	OUT State *	Current	OUT/TOUT status	LED
NPN (sinking)	OUTLVL0 (default)	OUT1	Sinking current	1	ON
NPN	OUTLVL0 (default)	OUT0	No current flow	0	OFF
NPN	OUTLVL1	OUT1	No current flow	1	OFF
NPN	OUTLVL1	OUT0	Sinking current	0	ON
PNP (sourcing)	OUTLVL0	OUT1	No current flow	1	OFF
PNP	OUTLVL0	OUT0	Sourcing current	0	ON
PNP	OUTLVL1 (default)	OUT1	Sourcing current	1	ON
PNP	OUTLVL1 (default)	OUT0	No current flow	0	OFF

* The output is “active” when it is commanded by the OUT, OUTP, or POUT command (for example, OUTxx1 activates output 3).

“General Purpose” (OUTFNCi-A)

The default function for the outputs is *General Purpose*. As such, the output is used as a standard output, turning it on or off with the OUT, OUTP or OUTALL commands to affect processes external to the controller. 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.

**Moving/Not Moving
(In Position)**
(OUTFNCi-<a>B)

When assigned the *Moving/Not Moving* function, the output will activate when the axis is commanded to move. As soon as the move is completed, the output will change to the opposite state.

Servo Axes: If the target zone mode is enabled (STRGTE1), the output will not change state until the move completion criteria set with the STRGTD and STRGTV commands has been met. (For more information, see the *Target Zone* section on page 74.) In this manner, the Moving/Not Moving output functions as an *In Position* output.

Example

The code example below defines onboard outputs 1 and 2 as *General Purpose* outputs and output 3 as a *Moving/Not Moving* output. Before the motor moves 4,000 steps, output 1 turns on and output 2 turns off. These outputs remain in this state until the move is completed, then output 1 turns off and output 2 turns on. While the motor/load is moving, output 3 remains on.

```
SCALE0      ; Disable scaling
MC0         ; Set axis 1 to preset positioning mode
MA0        ; Select incremental positioning mode
A10        ; Set axis 1 acceleration to 10
V5         ; Set axis 1 velocity to 5
D4000      ; Set axis 1 distance to 4,000 counts
OUTFNC1-A   ; Set onboard output 1 as a general purpose output
OUTFNC2-A   ; Set onboard output 2 as a general purpose output
OUTFNC3-1B  ; Set onboard output 3 as axis 1 Moving/Not Moving output
OUT10      ; Turn onboard output 1 on and output 2 off
GO1        ; Initiates axis 1 move
OUT01      ; Turn onboard output 1 off and output 2 on
```

**Program in
Progress**
(OUTFNCi-C)

When assigned the *Program in Progress* function, the output will activate when a program is being executed. After the program is finished, the output's state is reversed. The action of executing a program is also reported with system status bit 3 (see TSSF, TSS and SS commands).

Limit Encountered
(OUTFNCi-<a>D)

When assigned the *Limit Encountered* function, the output will activate when a hard or soft end-of-travel limit has been encountered.

If a hard or soft limit is encountered, you will not be able to move the motor/load 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 LHØ command, but this is recommended only if the motor is not coupled to the load.) The event of encountering an end-of-travel limit is also reported with axis status bits 15-18 (see TASF, TAS and AS commands, summary on page 226).

Stall Indicator
(OUTFNCi-<a>E)
Stepper Axis Only

When assigned the *Stall Indicator* function, the output will activate when a stall is detected. To detect a stall, you must first connect an encoder and enable stall detection with the ESTALL1 command. Refer to *Encoder-Based Stepper Operation* on page 64 for further discussion on stall detection.

Fault Output
(OUTFNCi-F)

When assigned the *Fault Output* function, the output will activate 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 LIMFNCi-F or INFNCi-F command (see page 83).

Make sure the drive fault input is enabled (DRFEN) and the drive fault active level (DRFLVL) is appropriate for the drive you are using.

**Maximum Position
Error Exceeded**
(OUTFNCi-<a>G)
Servo Axes Only

When assigned the *Maximum Position Error Exceeded* function, the output will activate when the maximum allowable position error, as defined with the SMPER command, is exceeded.

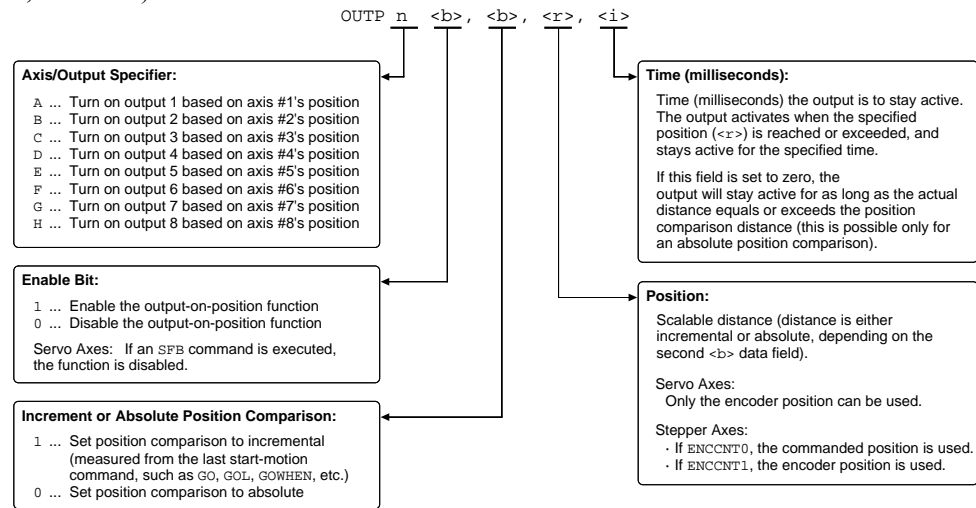
The position error (TPER) is defined as the difference between the commanded position (TPC) and the actual position as measured by the feedback device (TFB). When the maximum position error is exceeded (usually due to lagging load, instability, or loss of position feedback), the controller shuts down the drive and sets error status bit 12 (reported by the TERF, TER and ER commands if bit 12 of the ERROR command is enabled).

NOTE

If the SMPER command is set to zero (SMPERØ — the default value), the position error is not monitored; thus, the *Maximum Position Error Exceeded* function will not be usable.

Output on Position (OUTFNCi-H)

The *Output on Position* feature activates the designated output when the axis has reached a specified position. To use this feature, you must first assign the Output on Position function to the respective output, and define the Output on Position characteristics with the `OUTP` command. The `OUTP` command correlates directly to a specific onboard programmable output and axis (e.g., `OUTPA` correlates to output 1 and axis 1, `OUTPB` correlates to output 2 and axis 2, and so on):



Sample Code for Setup

In this example, the user has a two-axis application and wants to turn on output when axis 1 (servo) reaches encoder position 50,000 and turn on another output when axis 2 (stepper) reaches commanded position 30,000. Both outputs must remain on for 50 milliseconds.

```
AXSDEF10           ; Define axis 1 as servo, axis 2 as stepper
SFB1              ; Select encoder feedback for axis 1
OUTFNC1-H         ; Define onboard output 1 as an "output on
                  ; position" output
OUTFNC2-H         ; Define 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)
```

Notes about Output On Position

- On servo axes, this feature can be used only with encoder feedback and is not operational with ANI (analog input) feedback.
On stepper axes, this feature can be based on commanded position or encoder position, depending on the `ENCCNT` setting for the particular axis (default is `ENCCNT0`, which uses the commanded position).
- The output activates only during motion; thus, 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.

Variable Arrays (*teaching* variable data)

More on variables:
see page 18.

Variable data arrays provide a method of storing (*teaching*) variable data and later using the stored data as a source for motion program parameters. The variable data can be any value that can be stored in a numeric (VAR or VARI) variable (e.g., position, acceleration, velocity, etc.). The variable data is stored into a *data program*, which is an array of *data elements* that have a specific address from which to write and read the variable data. Data programs do not contain 6K Series commands.

The information below describes the principles of using the data program in a teach-type application. Following that is an application example in which the joystick is used to teach position data to be used in a motion program.

Basics of Teach-Data Applications

The basic process of using a data program for data teaching applications is as follows:

1. Initialize a data program.
2. Teach (store/write) variable data into the data program.
3. Read the data elements from the data program into a motion program.

1. Initialize a Data Program

This is accomplished with the DATSIZ command. The DATSIZ command syntax is DATSIZ*i*<, *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 (DATP*i*). 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 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 DATP*i* 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=+0.0, +0.0, +0.0, +0.0
*DATA=+0.0, +0.0, +0.0, +0.0
*DATA=+0.0, +0.0, +0.0, +0.0
*DATA=+0.0
```

Each data statement, comprising four data elements, uses 43 bytes of memory. The memory for each data statement is subtracted from the memory allocated for user programs (see MEMORY command).

2. Teach the Data to the Data Program

The data that you wish to write to the data elements in the data program must first be placed into numeric variables (VAR). Once the data is stored into numeric variables, the data elements in the data program can be edited by using the Data Pointer (DATPTR) command to move the data pointer to that element, and then using the Data Teach (DATTC) command to write the datum from the numeric variable into the element.

When the DATSIZ command is issued, the internal data pointer is automatically positioned to data element 1. Using the default settings for the DATPTR command, the numeric variable data is written to the data elements in sequential order, incrementing one by one. When the last data element in the data program is written, the data pointer is automatically set to data element 1 and a warning message (*WARNING: POINTER HAS WRAPPED AROUND TO DATA POINT 1) is displayed. The warning message does not interrupt program execution.

The DATPTR command syntax is `DATPTRi, i, i`. The first integer (i) represents the data program number (1 through 50). The second integer represents the number of the data element to point to (1 through 6500). The third integer represents the number of data elements by which the pointer will increment after writing each data element from the DATTC command, or after recalling a data element with the DAT command.

The DATTC command syntax is `DATTCi<, i, i, i>`. Each integer (i) represents the number of a numeric variable. The value of the numeric variable will be stored into the data element(s) of the currently active data program (i.e., the program last specified with the last DATSIZ or DATPTR command). As indicated by the number of integers in the syntax, the maximum number of variable values that can be stored in the data program per DATTC command is 4. Each successive value from the DATTC command is stored to the data program according to the pattern established by the third integer of the DATPTR command.

As an example, suppose data program 1 is configured to hold 13 data elements (DATSIZ1, 13), the data pointer is configured to start at data element 1 and increment 1 data element after every value stored from the DATTC command (DATPTR1, 1, 1), and the values of numeric variables 1 through 3 are already assigned (VAR1=2, VAR2=4, VAR3=8). If you then enter the DATTC1, 2, 3 command, the values of VAR1 through VAR3 will be assigned respectively to the first three data elements in the data program, leaving the pointer pointing to data element 4. The response to the TPROG DATP1 command would be as follows (the text is highlighted to illustrate the final location of the data pointer after the DATTC1, 2, 3 command is executed):

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

If you had set the DATPTR command to increment 2 data elements after every value from the DATTC command (DATPTR1, 1, 2), the data program would be filled differently and the data pointer would end up pointing to data element 7:

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

3. Recall the Data from the Data Program

After storing (*teaching*) your variables to the data program, you can use the DATPTR command to point to the data elements and the DATi (“i” = data program number) data assignment command to read the stored variables to your motion program. *You cannot recall more than one data element at a time; therefore, if you want to recall the data in a one-by-one sequence, the third integer of the DATPTR command must be a 1 (this is the default setting).*

Summary of Related 6K Series Commands

- DATSIZ Establishes the number of data elements a specific data program is to contain. A new DATP_i program name is automatically generated according to the number of the data program (*i* = 1 through 50). The memory required for the data program is subtracted from the memory allocated for user programs (see MEMORY command).
- DATPTR Moves the data pointer to a specific data element in any data program. This command also establishes the number of data elements by which the pointer increments after writing each data element from the DATTCB command and after recalling each data element with the DAT command.
- DATTCB Stores the variable data into the data program specified with the last DATSIZ or DATPTR command. After the data is stored, the data pointer is incremented the number of times entered in the third integer of the DATPTR command. *The data must first be assigned to a numeric variable before it can be taught to the data program.*
- TDPTR Responds with a 3-integer status report (*i*, *i*, *i*): First integer is the number of the active data program (the program specified with the last DATSIZ or DATPTR command); Second integer is the location number of the data element to which the data pointer is currently pointing; Third integer is the increment set with the last DATPTR command.
- [DPTR] ... From the currently active data program, uses the number of the data pointer's location in a numeric variable assignment operation or a conditional statement operation.
- [DATP_i] . The name of the data program created after issuing the DATSIZ command. The integer (*i*) represents the number of the data program. Data programs can be deleted just like any other user program (e.g., DEL DATP1).
- [DAT_i] ... From the data program specified with *i*, assigns the numeric value of the data element (currently pointed to by the data pointer) to a specified variable parameter in a 6K series command (e.g., D (DAT3) , (DAT3)).

Teach-Data Application Example

In this example, 2 axes (axis 1 and axis 2) of a 6K Series controller are used to move a 2-axis stage. This example illustrates a common method of teaching a path by using the joystick to move the load into position, teach the position (triggered by the Joystick Release input), then move to the next position. Five positions will be taught from each axis (2 axes at one trigger), for a total of 10 data elements in the data program. After all 10 positions are taught to the data program, the controller will automatically move both axes to a home position, move to each position that was taught, and then return to the home position.

For the sake of brevity, this example is limited to teaching 10 position data points; however, in a typical application, many more points would be taught. Also, it is assumed that end-of-travel and home limits are wired and a homing move has been programmed.

What follows is a suggested method of programming the controller for this application. To accomplish the teach application, a program called MAIN is created, comprising three subroutines: SETUP (to set up for teaching data to the data program), TEACH (to teach the positions), and DOPATH (to implement a motion program based on the positions taught).

The joystick operation in this example is based on using an analog inputs (ANI) SIM on an expansion I/O brick. The ANI SIM is in slot 2 of I/O brick 1 and inputs 1 and 2 are used to control axes 1 and 2, respectively. A digital input SIM is installed in slot 1 of I/O brick 1, and input 1 on that SIM is assigned the Joystick Release function to trigger the position teach operation.

Step 1 Initialize a Data Program.

```
DEL DATP1          ; Delete data program 1 (DATP1) in preparation for
                  ; creating a new data program 1
DATSIZ1,10        ; Create data program 1 (named DATP1) with an
                  ; allocation of 10 data elements. Each element is
                  ; initialized to zero.
```

Step 2 Define the SETUP Subroutine. The SETUP subroutine need only run once.

```
DEF SETUP          ; Begin definition of the subroutine called SETUP
1INFNC1-M         ; Assign digital input 1 on SIM 2 (I/O point 1) of
                  ; I/O brick 1 to function as a "Joystick Release" input.
JOYVH3,3          ; Set the maximum velocity (3 units/sec) that axes 1
                  ; and 2 can achieve when the joystick is at full
                  ; deflection
JOYAXH1-9,1-10    ; Assign ANI input 1 on SIM 2 (I/O point 9) of I/O
                  ; brick 1 to control axis 1; assign ANI input 2 on
                  ; SIM 2 (I/O point 10) of I/O brick 1 to control
                  ; axis 1
VAR1=0            ; Initialize variable 1 equal to zero
VAR2=0            ; Initialize variable 2 equal to zero
DRIVE11          ; Enable the drives for both axes
MA11             ; Enable the absolute positioning mode for both axes
END              ; End definition of the subroutine called SETUP
```

Step 3 Define the TEACH Subroutine.

```
DEF TEACH          ; Begin definition of the subroutine called TEACH
HOM11             ; Home both axes (absolute position counter is set to
                  ; zero after the homing move)
DATPTR1,1,1       ; Select data program 1 (DATP1) as the current active
                  ; data program, and move the data pointer to the first
                  ; data element. After each DATTCH value is stored to
                  ; DATP1, increment the data pointer by 1 data element.
REPEAT           ; Set up a repeat/until loop
JOY11            ; Enable joystick mode on both axes. At this point,
                  ; you can start moving the axes into position with the
                  ; joystick. WHILE USING THE JOYSTICK, COMMAND PROCESSING
                  ; IS STOPPED HERE UNTIL YOU ACTIVATE THE JOYSTICK
                  ; RELEASE INPUT. Activating the joystick release input
                  ; disables the joystick mode and allows the subsequent
                  ; commands to be executed (assign the current positions
                  ; to the variables and then store the positions in the
                  ; data program).
VAR1=1PM         ; Set variable 1 equal to the position of motor 1
VAR2=2PM         ; Set variable 2 equal to the position of motor 2
DATTCH1,2        ; Store variable 1 and variable 2 into consecutive
                  ; data elements. (The first time through the
                  ; repeat/until
                  ; loop, variable 1 is stored into data element 1 and
                  ; variable 2 is stored into data element 2. The data
                  ; pointer is automatically incremented once after each
                  ; data element and ends up pointing to the third data
                  ; element in anticipation of the next DATTCH command.)
WAIT(1IN.1=b1)   ; Wait for the joystick release input to be de-activated
UNTIL(DPTR=1)    ; Repeat the loop until the data pointer wraps around
                  ; to data element 1 (data program full)
END              ; End definition of the subroutine called TEACH
```

Step 4 Define the DOPATH Subroutine.

```
DEF DOPATH          ; Begin definition of the subroutine called DOPATH
HOM11              ; Move both axes to the home position
                  ; (absolute counters set to zero)
A50,50            ; Set up the acceleration
V3,3              ; Set up the velocity
DATPTR1,1,1       ; Select data program 1 (DATP1) as the current active
                  ; data program, and set the data pointer to the first
                  ; data element. Increment the data pointer one element
                  ; after every data assignment with the DAT command.
                  ; If you wanted to move only axis 1 down the taught
                  ; path, you would set the increment (third integer)
                  ; to a 2, thus accessing only the axis 1 stored
                  ; positions.
REPEAT            ; Set up a repeat/until loop
D(DAT1), (DAT1)   ; The position of axis 1 and axis 2 are recalled into
                  ; the distance command
GO11              ; Move to the position
T.5               ; Wait for 0.5 seconds
UNTIL (DPTR=1)    ; Repeat the loop until the data pointer wraps around
                  ; to data element 1 (all data elements have been read)
HOM11             ; Move both axes back to the home position
END               ; End definition of the subroutine called DOPATH
```

Step 5 Define the MAIN Program (Include SETUP, TEACH, and DOPATH).

```
DEF MAIN          ; Begin definition of the program called MAIN
SETUP             ; Execute the subroutine called SETUP
TEACH             ; Execute the subroutine called TEACH
DOPATH           ; Execute the subroutine called DOPATH
END              ; End definition of the program called MAIN
```

Step 6 Run the MAIN Program and Teach the Positions with the Joystick.

1. Enter the MAIN command to execute the teach application program and set the joystick's *axis select* input to high.
2. Use the joystick to move to the position to be taught.
3. Once in position, activate the *joystick release* input to teach the positions. Two positions (one for each axis) are taught each time you activate the joystick release input.
4. Repeat steps 2 and 3 for the remaining four teach locations. After triggering the joystick release input the fifth time, the controller will home the axes, repeat the path that was taught, and then return both axes to the home position.